PROGRAMMEZ!

Le magazine des développeurs

100% CLOUD COMPUTING

Serverless, Terraform
OpenTelemetry
GCP, Azure, AWS
IoT & cloud
CI/CD en mode cloud

CI/CD en mode cloud
IoT & cloud
GCP, Azure, AWS
OpenTelemetry
Serverless, Terraform
100% CLOUD COMPUTING

M 01642 - 10 - F: 6,99 € - RD

SPÉCIAL HIVER 2023

Numéro déployé avec

metanext / vmware® OUTSCALE OVHcloud NumSpot Cloud. Européen. Citoyen

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

Précommandez dès aujourd'hui le volume 4 de notre Histoire de la micro-informatique !

Ce volume est consacré à la mirco-informatique française.

PARTICIPEZ À LA CAMPAGNE ULULE :

https://fr.ulule.com/histoire-de-la-micro-informatique-volume-4-30-ans-de-micro-informatique-francaise/

Fin : le 21 mars



#HS10

IaaS ou PaaS : un débat dépassé !

L'intérêt des services managés pour les entreprises de toute taille ne fait plus débat, et ils sont de plus en plus populaires auprès de celles qui cherchent à se libérer des tâches fastidieuses liées à la gestion de leur infrastructure.

Si pour beaucoup de développeurs au sein d'entreprises, ce sujet est celui de leurs collègues « des opérations » ou de « l'équipe plateforme », il semble intéressant pour tous de considérer les possibles d'aujourd'hui. Il y a encore 5 ans, il était fréquent d'opposer les approches « Infrastructure as a service » et « Platform as a service ». L'adage veut que la première ait l'avantage du coût à l'échelle et de la liberté d'arbitrer entre plusieurs fournisseurs, tandis que la seconde augmente la productivité des équipes, en acceptant l'adhérence (ou « lock-in ») liée à un acteur spécifique. On opposait à l'époque la productivité des équipes de développement à l'optimisation des coûts et de la liberté des équipes en charge de la production.

Mais technologies et solutions du marché ont bien évolué et il est temps de reconsidérer cette vue dépassée. L'âge d'or d'Heroku est loin derrière nous et aucun acteur n'a su depuis imposer une plateforme réversible basée sur un écosystème « tout inclus ». Et c'est parce qu'on a goûté les limites de cette approche « opinionated » comme disent nos amis anglophones, que l'on sait qu'il ne s'agit pas d'une panacée. Il est sage de la réserver à des équipes métiers pour les projets les moins sensibles ou durables.

À l'heure des conflits internationaux et du RGPD, réversibilité et approche multicloud sont devenues des impératifs pour les applicatifs cœur du business. La souveraineté est un sujet d'aujourd'hui pour beaucoup, et pour les autres un prérequis pour attaquer les marchés de demain (Chine, secteurs stratégiques et sensibles…).

Heureusement, un autre chemin s'est dégagé et l'on a même élu ses champions ! Depuis le début de la décennie, ce sont les conteneurs et Kubernetes pour le calcul, ainsi qu'une dizaine de systèmes de base de données (DBMS) open source, associés au stockage objet. Ces technologies ubiquitaires permettent de couvrir la quasi-totalité des enjeux pour n'importe quelle application moderne, sur site comme dans le cloud, que son déploiement soit mondialisé ou souverain.

Kubernetes, comme les « PaaS », facilite le déploiement, la gestion (autohealing notamment) et de l'évolution efficace des applications. Il offre une grande flexibilité, tout en garantissant la réversibilité grâce à une API standard et commune (les fournisseurs de cloud de toute taille offrent ce service, et de nombreux éditeurs offrent des solutions sur site).

Les bases de données managées (ou DBaaS), offrent également de nombreux avantages. Si PostgreSQL se positionne comme la base de données « tout terrain », d'autres standards ont émergé pour des données plus spécifiques (M3DB pour les séries temporelles, Elastic ou Opensearch pour les données textuelles…)

En combinant ces technologies au catalogue de tous les fournisseurs cloud, les entreprises peuvent libérer leurs équipes de la gestion de leur infrastructure, tout en leur offrant les outils nécessaires pour développer efficacement leurs applications. Cette combinaison met fin au débat entre IaaS et PaaS, en offrant les avantages de chacun des deux modèles d'hier. Le stockage objet a d'ailleurs été précurseur de cette nouvelle frontière floue entre infrastructure de base et service géré, désormais lui aussi accessible au travers d'un protocole et API communs à la plupart des fournisseurs.

> ***« IaaS ou PaaS, dépassé est le débat »***
> ***(Légende de la Force)***

En les consommant « en tant que service », ces nouveaux piliers de la plupart des « stack » modernes permettent de gérer efficacement les données, tout en éliminant les tâches fastidieuses liées à l'installation, à la configuration et à la gestion des machines physiques ou virtuelles. Ces technologies permettent également de faciliter les aspects tels que la reproductibilité et la mise à l'échelle, tout en garantissant la réversibilité. Elles encouragent les approches « devops » et la responsabilité bout-en-bout sur un périmètre applicatif donné, de la conception aux opérations. Des équipes logicielles multi-compétentes peuvent ainsi se concentrer sur le développement de leurs applications plutôt que sur la gestion de leur infrastructure : la promesse du PaaS d'hier sans l'attachement à certains langages et à un fournisseur qu'il sous-tendait.



@Maxime Hurtrel
Group Product Manager chez OVHcloud

PROCHAIN NUMÉRO

PROGRAMMEZ! N°257

Disponible à partir du
28 avril 2023

NUMÉRO EN COURS

PROGRAMMEZ! N°256

Disponible depuis le
3 février 2023

Contenus

3 Edito
IaaS ou Pass : un début dépassé !
Maxime Hurtrel

5 & 7 Abonnement
Les partenaires 2023 de Programmez!

6 Agenda
Toutes les conférences dévs à venir

8 Cloud tu veux, cloud tu auras
François Tonic

10 Numspot : un cloud certifié et souverain
François Tonic

11 Cloud souverain : une étape essentielle
Laurent Allard

12 Database en tant que service : quels sont les choix ?
Thierry Bollet

15 Cockpit : la plateforme de gestion d'OUTSCALE

18 Gérer facilement son infra avec le provider Terraform OVHcloud
Aurélie Vache

22 IaC avec les solutions OVHcloud - partie 1
Olivier Beautier

26 Heroku payant : quelles alternatives pour déployer ses PoC ?
Antoine Garnier & Paul Barrié

31 L'avenir de l'IA est-il dans le cloud ?
Kevin Ansard

35 Serverless : comment en tirer parti ?
Simon Shillaker

38 Les fonctions serverless, c'est pour tout le monde !
Sébastien Stormacq

42 Développer, déployer et sécuriser les accès aux ressources (cloud) grâce à Azure AD
Clément Sannier

46 Cloud : un Idem pour les gouverner tous
David Jakubowicz

48 Une app de gestion EPUB en mode Cloud
Lionel Dimitrijevic

52 Intégration et livraison contenues sur GCP partie 1
Laurent Grangeau

56 Monitorer la santé des apps avec OpenTelemetry
Brice Dekany

59 Développer un module Azure IoT Edge en C#
Kevin Beaugrand

62 Google Cloud Armor & Cloud CDN : configuration & tips
Tanguy Baudrin & Nidal el Haddad

67 Estimez vos coûts Azure avec Infracost + Terraform
Mikael Krief

70 Code Space : propulsez votre app Angular dans le cloud !
Pierre Bouillon

74 Azure Spring Apps Enterprise
Steve P. & Fabien Martin

79 Cloud Seed : déployer plus rapidement avec GitLab
Jean-Philippe Baconnais

Abonnement numérique (format PDF)
directement sur www.programmez.com

L'abonnement à Programmez! est de 55 € pour 1 an, 90 € pour 2 ans.
Abonnements et boutiques en pages 42 et 25

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

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

PROGRAMMEZ!

Le magazine des développeurs

nos classiques

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

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

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

Option : accès aux archives — **20€**

** Tarifs France métropolitaine*

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

abonnement numérique

PDF ..... **45€**
1 an

Souscription directement sur
www.programmez.com

Offre 2023

Notre partenaire be quiet! vous offre le Silent Wings 4 pour tout abonnement 1 ou 2 ans

Silent Wings 4 un ventilateur haut de gamme pour PC. Il permettra de refroidir vos radiateurs et dissipateurs. Un bouton permet de régler la vitesse.

Modèle 120mm PWM. Il est livré complet avec son kit de fixation et le ventilateur. Prix public : 23,90 €

be quiet! est une marque haut de gamme d'alimentations, de solutions de refroidissement et de boîtiers PC.



1 an Programmez! 61€
+ Silent Wings 4 de be quiet

2 ans Programmez! 96€
+ Silent Wings 4 de be quiet

Quantité limitée. Cette offre peut s'arrêter à tout moment.
Tarif abonnement + 6 € de frais postaux. Cette offre n'est pas valable pour les abonnements étudiants et PDF. Sous réserve d'erreur typographique.
Délai moyen pour l'envoi : 7 jours

Toutes nos offres sur www.programmez.com

Oui, je m'abonne

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

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

- ☐ 1 an Programmez! : 61 €
  + Silent Wings 4 de be quiet
- ☐ 2 ans Programmez! : 96 €
  + Silent Wings 4 de be quiet

☐ Mme ☐ M. Entreprise : Fonction :

Prénom : Nom :

Adresse :

Code postal : Ville :

Adresse email indispensable pour la gestion de votre abonnement

E-mail : @

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

*** Tarifs France métropolitaine**

Offres pouvant s'arrêter à tout moment, sans préavis **PROG spécial 10**

Les événements Programmez!

Meetups Programmez!

Les meetups 2023 :

28 mars
25 avril
16 mai
27 juin

Tous les détails et les thèmes des sessions sur programmez.com
Où : Theodo 48 BD des Batignolles, Paris
A partir de 18h30

Conférence DevCon #19

27 avril
à Lyon

Informations & inscription : programmez.com

mars 2023

Lun.	Mar.	Mer.	jeu.	Ven.	Sam.	Dim.
		1	2	3	4	5
6	7 Kubernetes Community Days France / Paris	8	9 MDF 2023 / Paris	10	11	12
13	14	15	16	17	18	19
20	21	22	23 SymfonyLive Paris Agile Niort	24	25	26
27	28 Meetup Programmez!	29	30	31		

avril 2023

Lun.	Mar.	Mer.	jeu.	Ven.	Sam.	Dim.
					1 JDLL / Lyon	2
3	4	5 FIC / Lille	6	7	8	9
10	11	12 Devoxx / Paris	13 Mixit / Lyon	14	15	16
17	18	19	20 Toulouse Hacking Convention	21	22	23
24	25 Meetup Programmez!	26	27	28	29	30

2023

- AFUP Day Lille & Lyon : 12 mai
- Newcrafts Paris : 25 & 26 mai
- DevFest Lille : 26 mai
- Polycloud : 27 mai / Montpellier
- Serverless Days Paris : 7 juin
- Sunny Tech : 29 & 30 juin / Montpellier
- Best of web : juin / Paris
- Riviera DEV : 10-12 juillet / Sophia Antipolis
- Salon de la Data : 19 septembre / Nantes
- API Platform conference : 21 & 22 septembre / Lille

Merci à Aurélie Vache pour la liste 2022-23, consultable sur son GitHub : https://github.com/scraly/developers-conferences-agenda/blob/master/README.md

PROGRAMMEZ!
Directives de compilation

Programmez! hors-série n°10 - HIVER 2022-2023
Directeur de la publication & rédacteur en chef
François Tonic - ftonic@programmez.com
Code review : **Dorra Bartaguiz**
Contacter la rédaction : redaction@programmez.com

Les contributeurs techniques
Maxime Hurtrel, Laurent Allard, Thierry Bollet, Aurélie Vache, Olivier Beautier, Antoine Garnier, Paul Barrié, Simon Shillaker, Sébastien Stormacq, Clément Sannier, David Jakubowicz, Anthony Chateauneuf, Yacine Boulbot, Laurent Grangeau, Brice Dekany, Kevin Beaugrand, Tanguy Baudrin, Nida el Haddad, Mikael Krief, Pierre Bouillon, Steve P., Fabien Martin, Jean-Philippe Baconnais, Lionel Dimitrijevic

Maquette : **Pierre Sandré**

Marketing – promotion des ventes
Agence BOCONSEIL
Analyse Media Etude
Directeur : Otto BORSCHA
oborscha@boconseilame.fr
Responsable titre : Terry MATTARD
Téléphone : 09 67 32 09 34

Publicité
Nefer-IT - Tél. : 09 86 73 61 08
ftonic@programmez.com

Impression : **SIB Imprimerie, France**
Dépôt légal : **A parution**

Commission paritaire
1225K78366

ISSN : **2279-5001**

Abonnement
Abonnement (tarifs France) : 55 € pour 1 an, 90 € pour 2 ans. Etudiants : 45 €. Europe et Suisse : 65 € - Algérie, Maroc, Tunisie : 64 € - Canada : 80 € - Tom - Dom : voir www.programmez.com.
Autres pays : consultez les tarifs sur www.programmez.com.

Pour toute question sur l'abonnement :
abonnements@programmez.com

Abonnement PDF
monde entier : 45 € pour 1 an.
Accès aux archives : 20 €.

Nefer-IT
57 rue de Gisors, 95300 Pontoise France
redaction@programmez.com
Tél. : 09 86 73 61 08

Toute reproduction intégrale ou partielle est interdite sans accord des auteurs et du directeur de la publication.
© Nefer-IT / Programmez!, mars 2023.

Les partenaires 2023 de

PROGRAMMEZ!

Le magazine des dévs



Niveau maître Jedi

salesforce

Progress®

Scaleway

The cloud of choice

Niveau padawan

OSAXiS



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

Contactez-nous dès maintenant :

ftonic@programmez.com



François Tonic
Rédacteur en chef de Programmez !

Cloud tu veux, cloud tu auras !

Depuis 15 ans, le cloud computing est incontournable dans l'informatique, le développement et les multiples services et apps que nous utilisons au quotidien. Ils sont tellement dans notre routine que l'on oublie parfois qu'il s'agit de cloud computing. Gmail, Netflix, Office 365, iCloud, Google Drive, Dropbox, les jeux mobiles, etc. Derrière, nous trouvons naturellement d'énormes infrastructures réseau et serveurs. Eh oui, nous oublions que le cloud a beau être quelque part dans le « nuage », il fonctionne forcément sur des serveurs, du stockage, des CPU et GPU. Ce n'est pas magique et uniquement du virtuel. Nous sommes passés des logiciels physiques sur nos terminaux avec des supports physiques à des apps dématérialisées que nous installons depuis des portails et des App stores.

Pour faire fonctionner tous les services de cloud computing que nous utilisons, que nous soyons dévs, utilisateurs, ops, il faut déployer des baies contenant les serveurs et le stockage. Prenons un exemple. En 2020, OpenAI, qui a créé ChatGPT, commençait à utiliser une infrastructure Azure : +285 000 cœurs CPU et +10 000 GPU, des switchs réseaux 400 Gb/s pour assurer une communication la plus performante entre les serveurs et les baies. Netflix repose sur les infrastructures AWS. Le service vidéo repose sur 20 000 serveurs physiques et plus de 100 000 instances.

Bref, tous les services de vidéo à la demande, les apps, la migration des serveurs physiques vers le cloud, l'usage de terminaux de type Chromebook, de jeux streamés, consomment de plus en plus de ressources cloud d'où la construction continue de nouveaux datacenters et la modernisation de ceux existants. CQFD. Le cloud a changé notre approche de l'usage de l'informatique et des technologies, mais il n'a fait que déplacer le matériel des entreprises et chez soi vers les datacenters.

Hyperscalers

Ce que nous appelons les hyperscalers ce sont les fournisseurs de cloud public qui sont déployés à travers le monde. Ils disposent d'un catalogue de plusieurs centaines de services cloud.

Aujourd'hui, et plus que jamais, trois fournisseurs se partagent le marché :

1er : AWS (Amazon)
2e : Azure (Microsoft)
3e : Google Cloud Platform (Google)

Depuis 3 ans, Microsoft réussit à imposer son cloud public Azure en multipliant les services et notamment tout ce qui est Office 365. En 2022, au niveau mondial, Azure dépassait, selon Synergy Research Group, les 20 % de part de marché. AWS reste solide leader avec 32-33 %. Google a reboosté sa plateforme cloud (GCP) et GCP grignote trimestre après trimestre sur les concurrents. Au-delà des 3 géants, nous trouvons Alibaba qui bénéficie de l'immense marché chinois et asiatique (environ 6 %) et Kyndryl (IBM) qui peine à exister. La baisse est régulière depuis 2017.

En France, la tendance n'est pas différente : AWS domine assez largement le marché. Azure et GCP suivent, à bonne distance. Oracle est encore plus loin. Le marché est très segmenté au-delà avec 30 % du marché captés par d'autres cloud publics et notamment par les fournisseurs français tels que OVHcloud, OutScale, Scaleway et des intégrateurs tels que Atos, Cap Gemini, etc.

Les fournisseurs ont beaucoup segmenté les offres pour répondre aux usages et problématiques des entreprises. Par exemple, on dispose de solutions complètes pour gérer et déployer une IA, des IoT ou pour mettre en place des usages par secteur d'activité. Ces verticales par activité rassemblent divers services, existants en dehors et/ou spécifiquement développés.

Plusieurs clouds

Rappelons que sous le terme générique de cloud computing, nous trouvons plusieurs approches. Depuis 15 ans, le cloud est traditionnellement segmenté en 3 « fonctions » :

- IaaS : infrastructure en mode cloud typiquement des serveurs que l'on virtualise sur des instances IaaS d'un fournisseur. C'est le premier service à avoir connu le succès. Aujourd'hui, le IaaS reste demandé. Il est devenu un service de commodité
- PaaS : plateforme en mode cloud. Typiquement pour des bases de données managées, déployer des apps/apps Web.
- SaaS : tous les services que nous consommons tels que la bureautique, le stockage, le CRM, l'ERP, etc.

De ce triptyque originel, nous trouvons désormais le cloud privé managé, le cloud hybride, le multicloud. Aujourd'hui, on dispose aussi de services cloud, ou de clouds, pour la santé, avec des impératifs de sécurité et réglementaire strict ou des clouds pour les États.

La notion de multicloud s'impose depuis 3 ans. Il ne s'agit plus d'utiliser un unique fournisseur, mais de pouvoir combiner plusieurs clouds publics, ou privés et de les gérer depuis une console d'administration unifiée. Le multicloud permet de diversifier les fournisseurs technologiques, voire, de répartir les risques en cas de coupures réseaux ou d'une panne de grandes ampleurs.

Les promesses d'un monde plus ouvert ont parfois été peu tenues. Car finalement, si la promesse était d'être agnostique de tout fournisseur, rapidement, nous nous sommes rendu compte que nous restions souvent liés aux fournisseurs et qu'ils y avaient parfois des « verrous » plus ou moins explicites sur un service, par exemple, pour le stockage ou les bases de données. Bref l'adhérence technique continue à exister. Le multicloud, l'open source, et les standards doivent lever ces problèmes et améliorer la réversibilité.

Le cloud et la géographie

Les fournisseurs segmentent leur cloud par région. Une région correspond à une zone géographique. Par exemple, OVHcloud possède deux régions en France : Strasbourg et Gravelines. AWS déploie pour la région Europe plusieurs sites et chaque site correspond à des zones de disponibilités qui correspondent à un ou plusieurs datacenters : Milan, Londres, Paris, Espagne, etc. Lorsque vous provisionnez des ressources cloud, vous devez choisir la localisation région / zone de disponibilités. Ces régions sont importantes, car vous pouvez cibler les datacenters les plus proches de vos utilisateurs et améliorer ainsi les performances et la latence réseau.

Choisir c'est bien, comment choisir, c'est mieux

Comme toujours quand un dév doit coder un projet, utiliser des technologies, il doit se poser les bonnes questions, sauf si les technologies sont imposées de facto (ESN, entreprises, clients, le marché ciblé, etc.). Je ne vous apprends rien si je vous dis que pour développer une app pour Android je ne vais pas utiliser Python ou Swift. C'est une évidence, mais il ne faut pas l'oublier.

Dans le cloud, c'est un peu différent, du moins, pour un certain nombre d'usages et de fonctionnalités. Si vous êtes habitué aux environnements Windows/Microsoft, Azure pourra être un choix naturel. Si vous êtes dev Web, le choix sera bien plus large et là c'est plus une question d'affinité, de fonctionnalités, de coûts, de niveau de disponibilités (SLA).

Basiquement, nous pourrions mettre les critères suivants pour choisir un cloud :

- Les langages supportés
- Les frameworks et outils supportés (par exemple les CMS si vous faîtes surtout du Web)
- Réversibilité vs adhérence technique
- SLA et coûts
- Documentations, SDK, outils disponibles
- Intégration avec mon IDE

L'intégration transparente avec sa toolchain, et particulièrement son IDE, est un élément incontournable pour simplifier le CI/CD et le déploiement de son app. La disponibilité d'une CLI est aussi un autre critère critique surtout si vous êtes en charge de la gestion des ressources cloud.

Si vous intégrez des services cloud pour du compute vision, un traitement cloud depuis une app, le stockage, une base de données non locale, faire de l'IA, etc., la disponibilité des services est un point important sans oublier la disponibilité des SDK et des API avec le support du ou des langages utilisés.

Comprendre et définir votre projet, votre app, votre développement.

Avez-vous un service de type back sur votre app par exemple une SGBD, un service de stockage ? Le cloud back est très utilisé. Le choix est large et la plupart des fournisseurs de cloud auront peu ou prou le même choix de base de données et de stockage.

Quelques critères possibles :

1. quel type de base de données ai-je besoin : SQL, No SQL, non structurée, base documentaire, etc. À partir de là, vous pouvez comparer les offres de chaque fournisseur
2. Sur le stockage même chose : stockage simple, stockage bloc/objet, etc.
3. Où sont les datacenters ?
4. Tarifs et SLA
5. Possibilité de load balancer et/ou d'un CDN pour la latence

S'il s'agit de migrer des workloads d'un serveur local, ou d'un hébergeur Web, vers le cloud

Vous pouvez opter pour :

1. je migre tel quel mon workload typiquement sur une offre IaaS
2. je déploie uniquement la couche technique en PaaS avec un service cloud Web avec le même périmètre fonctionnel pour utiliser des services 100 % managés.
3. j'ai des Web apps de type JS/framework JS je peux utiliser un service PaaS de type Azure Web App
4. je regarde les tarifs, les SLA, la localisation des datacenters et éventuellement si j'ai besoin de performances les fonctions CDN, caching, etc.

Je veux utiliser une base de données en mode DBaaS

Le but est simple : je veux utiliser une base de données 100 % managée. Je vais donc utiliser une base de type DBaaS. Là encore, tous les fournisseurs proposent ces services. La différence se fera sur les bases disponibles et les types de bases (SQL, No SQL, etc.).

1. si j'utilise déjà une base de données dans mon app, je vais plutôt privilégier une base identique. Attention : si vous utilisez une ancienne version de la base, vous devrez peut-être mettre à niveau votre architecture et les modèles de données. En DBaaS, les versions supportées sont plus limitées. Si vous optez pour une approche IaaS, vous aurez plus de souplesse.
2. le choix du fournisseur peut être conditionné par le type de base utilisée ou votre historique. Si vous utilisez une base Oracle, Oracle Cloud sera peut-être un choix naturel, SQL Server je peux choisir naturellement Azure.
3. Là encore, regardez bien les tarifs, le SLA et tous les services annexes. Sur la tarification, regardez en détail tous les éléments tarifications pour éviter toute mauvaise surprise
4. si vous utilisez du No SQL, les principaux fournisseurs vont proposer les classiques Cassandra, MongoDB, Gremlin, etc. Vous pouvez éventuellement opter pour un DBaaS multimodèle

Dans tous les cas, si votre app existe, il faudra faire des ajustements d'architecture, remapper les couches de données.

Je veux utiliser des services cloud depuis mon code, comment faire ?

Aujourd'hui, il est possible d'intégrer/d'interagir avec les principaux services cloud grâce aux API et aux SDK des fournisseurs. Par exemple, si je suis développeur Java, je peux simplement intégrer des services Azure via le SDK Azure pour Java, idem sur AWS, GCP, etc.

Les principaux langages sont supportés et on dispose de SDK dédiés. Attention : tous les fournisseurs ne seront pas forcément au même niveau de support.

Notez aussi que pour les développeurs, les fournisseurs peuvent proposer des plans dédiés avec des quotas de ressources et/ou une limite en $ par mois. C'est un excellent point d'entrée pour utiliser les services de base.

N'oubliez pas de regarder le niveau de sécurité, les standards implémentés. La disponibilité d'une CLI devient un critère non négligeable. Je rappelle que la CLI permet d'administrer et de provisionner des services plus rapidement et vous aurez accès à l'ensemble des fonctions d'un service.

À vous de jouer !



François Tonic

NumSpot : un cloud certifié et souverain

L'écosystème cloud français n'est pas mort, bien au contraire. Depuis quelques mois, plusieurs annonces ont été faites pour renforcer les offres et de nouveaux acteurs font leur apparition. Vous ne connaissez pas NumSpot ? Faisons connaissance avec ce futur cloud 100% français.

L'ambition de NumSpot est très claire : proposer une plateforme cloud complète certifiée SecNumCloud (ANSSI) pour les entreprises, les administrations publiques et les collectivités et également une offre HDS pour le secteur de santé Les datacenters sont situés en France et les workloads et les données restent strictement en France. Il s'agit ici d'assurer une sécurité optimale des données les plus sensibles et respecter les réglementations françaises et européennes. Au-delà de SecNumCloud, NumSpot sera reconnu hébergeur de données de Santé (HDS). Les mots clés à retenir sont : protection des données, sécurité et souveraineté. Au-delà il s'agit de respecter les 3 types de souveraineté :

- Souveraineté technologie
- Souveraineté juridique
- Souveraineté des données

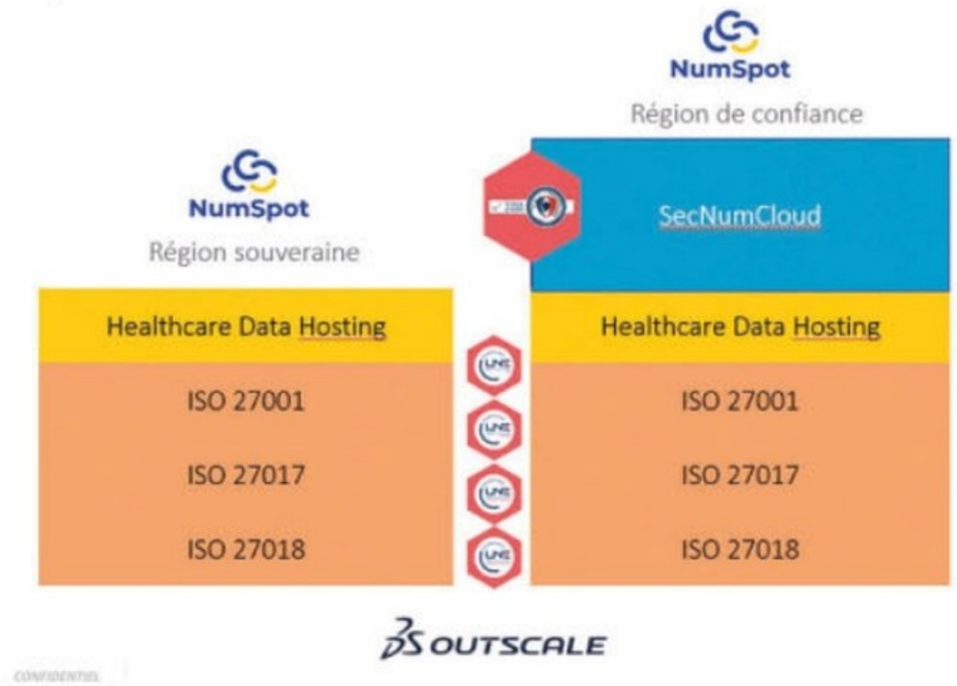
L'autre point clé de NumSpot concerne les actionnaires. Ils sont 100 % français : la Banque des Territoires, groupe Caisse des Dépôts, Docaposte, (filiale numérique du groupe La Poste) Dassault Systèmes via sa filiale Outscale et Bouygues Télécom. D'ailleurs, l'infrastructure de NumSpot s'appuiera sur les datacenters d'Outscale.

« Docaposte est le chef de file de ce projet ambitieux de cloud de confiance qui s'appuie sur la complémentarité de quatre grands acteurs français. Docaposte contribuera à la réussite de NUMSPOT par sa compréhension fine des attentes de la sphère publique et du secteur de la santé, par sa capacité à accompagner la transformation des organisations ainsi que par son expertise reconnue dans les services de confiance numérique. »
Olivier Vallet (président directeur général de Docaposte).

Les utilisateurs prioritaires

NumSpot, contrairement aux autres fournisseurs de cloud, veut se concentrer sur 4 cibles :

- Le secteur public
- Le monde de la santé
- L'assurance
- La banque/finance



Certains secteurs sont très en retard sur l'utilisation du cloud computing (IaaS, PaaS, SaaS). NumSpot proposera les briques techniques pour migrer en douceur, faire du cloud hybride, déployer des services 100 % cloud.

Un des points importants est d'être à l'abri des législations extraeuropéennes. NumSpot garantit ainsi les entreprises et les utilisateurs que les services clouds et toutes les données déposées dessus ne soient pas soumis à des lois et réglementations étrangères.

Une roadmap ambitieuse dès 2023

Dans un premier temps, NumSpot proposera à l'automne 2023, les premiers services autour du IaaS. Le IaaS est un service essentiel pour redéployer des applications, des bases de données sur le Cloud. Le IaaS NumSpot s'appuiera sur l'expertise et l'offre d'OutScale.

Au-delà, NumSpot veut proposer l'ensemble des couches Cloud que l'on s'attend à avoir d'un fournisseur : PaaS, CaaS, SaaS. Les conteneurs, Kubernetes, le Serverless sont des enjeux majeurs sur le cloud.

Vous l'aurez compris, l'objectif n'est pas de concurrencer les Azure, AWS ou GCP, mais de proposer une offre cohérente pour ceux qui sont attentifs à la sécurité et la confidentialité des données.

Techniquement, nous savons d'ores et déjà que l'ambition est de reposer sur un socle technique Open Source qui sera développé par les équipes NumSpot et les partenaires, tout en intégrant les standards cloud du marché. Les services développés seront certifiés pour une sécurité optimale.

L'autre objectif est de proposer une réversibilité : il sera possible de migrer les workloads sur un autre fournisseur cloud. Aucun verrou technique ne bloquera les migrations.

Les premiers recrutements sont en cours. Objectif : 50-60 personnes dans les prochains mois pour structurer l'offre et les premiers services. D'ici 12-18 mois, les équipes monteront à 200 personnes minimum.

SECNUMCLOUD : DÉFINITION

L'ANSSI définit le référentiel SecNumCloud. Il permet de qualifier des fournisseurs de cloud pour pouvoir offrir un catalogue d'éditeurs et de prestataires dits de confiance. Les critères du SecNumCloud sont mis à jour pour suivre les évolutions du marché. Par exemple, la version 3.2 précise les critères de protection vis-à-vis des lois extraeuropéennes. Cette version précise aussi la mise en œuvre de tests d'intrusion tout au long de la qualification pour tester la sécurité et la robustesse de l'offre.

Cloud souverain : une étape essentielle dans le développement du Cloud

Selon une étude récente réalisée par IDC pour VMware, 63% des entreprises européennes ont des données confidentielles (voire secrètes) dans le Cloud public. Les risques sur les données sont devenus la préoccupation dominante des DSI (ransomware, contrôle des metadata, protection des données, réglementations européennes, exposition à Cloud Act, …).



Laurent Allard

Leader Cloud Souverain VMware EMEA

Laurent dirige les efforts de VMware dans la région EMEA sur le cloud souverain/cloud de confiance. Il est membre de l'équipe de direction VMware EMEA. Laurent est également le moteur de l'implication de VMware dans Gaia-X.

Avant de rejoindre VMware il y a 3 ans, Laurent a dirigé des activités Tech chez AXA (CTO et co-fondateur d'AXA Tech), CGI (CTO Groupe), et OVH (CEO et Vice Chairman).

Pour entrer dans le sujet 'cloud souverain', commençons par nous entendre sur quelques définitions : 'cloud souverain' versus 'cloud de confiance' ; 'Souveraineté digitale' versus 'indépendance technologique'.

À ce jour, il n'existe pas de définition standard, ni de certification européenne, pour qualifier un cloud de "cloud souverain", ni même de terminologie commune "cloud souverain vs cloud de confiance". Cloud souverain et cloud de confiance sont des solutions. Mais quel est l'énoncé du problème : la souveraineté des données. C'est-à-dire l'ensemble des exigences clés associées aux données confidentielles des entreprises, telles que le contrôle des données et des métadonnées, la localisation physique, les niveaux de protection et l'exposition à une juridiction externe. Ces exigences critiques sur la souveraineté des données doivent se définir en premier, afin que les entreprises puissent ensuite garder le contrôle de leurs données en choisissant les plateformes appropriées pour héberger leurs données et innover de manière sécurisée.

Existe-t-il une réglementation européenne ?

Il n'y a pas de réglementation 'magique' au niveau européen (la situation géopolitique actuelle ne simplifie pas la définition d'une politique commune vis-à-vis des hyperscaleurs ; certains pays européens considèrent actuellement les USA comme un allié face à la tension armée et réduisent leur sensibilité à Cloud Act). Néanmoins, beaucoup de travaux intéressants vont harmoniser des critères d'exigence sur les solutions Cloud (les labels de Gaia-X ont été une première étape en 2022, et EUCS devrait aller plus loin courant 2023). Data Act va aussi renforcer le besoin de garder le contrôle de ses assets digitaux en forçant la portabilité (réversibilité).

L'Europe pousse à la 'Souveraineté digitale' (à ne pas confondre avec 'Indépendance industrielle européenne'). Le digital déployé en Europe repose à plus de 80% sur des composants matériel et logiciel fournis par des entreprises non européennes (Intel, AMD, Cisco, Dell, mais aussi AWS, Microsoft, Google …). Aucun scénario n'existe pour permettre une indépendance technologique de l'Europe. Mais il existe 2 types de technologies : celles qui reposent sur des standards (et les standards de fait), et d'autres qui reposent sur des architectures propriétaires (tel que certains PaaS qui créent des dépendances fortes : 'locking'). À défaut d'être indépendant, assurons-nous que nos choix permettent de garder notre autonomie / résilience : permettre de changer de fournisseur sans remettre en question les investissements applicatifs.

Définir une stratégie de Cloud souverain

La première étape est la classification de l'ensemble de ses données par niveau d'impact en cas de perte de contrôle (accès illicite, copie, blocage …). On peut considérer plusieurs niveaux d'impact :

- Niveau 4 : Atteinte exceptionnellement grave à l'intérêt national, aux entreprises, aux individus
- Niveau 3 : impact important
- Niveau 2 : impact moyen
- *Niveau 1 : impact limité*

Dans une stratégie de Cloud souverain, on cherchera à mettre les bons niveaux de protections pour chacun des 3 niveaux les plus critiques (4, 3 et 2). Le niveau 1 est en dehors des besoins de cloud souverain, car c'est le niveau standard que l'on peut héberger dans les clouds publics existants.

Pour chaque niveau de confidentialité, des critères d'exigence définissent les attributs du Cloud (certifications type SecNumCloud, exposition ou non au cloud Act, protection des metadata, localisation et citoyenneté des équipes d'exploitation, isolation des infrastructures, clés externes de chiffrement ….). Le travail d'analyse de l'ensemble des données peut être complexe (volumes importants, documentation limitée, notamment pour les données non-structurées). Il existe des outils pour faciliter la documentation des données existantes (NetApp Data Sense, ou Caveonix).

Le Multi-Cloud au cœur de la stratégie cloud souverain

Plusieurs niveaux de classification des données nécessitent plusieurs Clouds (multi-cloud). Chez VMware le même socle technique peut être déployé en mode dédié (mono-client), ou en Public cloud souverain (à travers un réseau étendu de partenaires souverains : https://cloud.vmware.com/providers/search-result.

L'évaluation d'un partenaire VMware comme étant 'Souverain' repose sur 20 points de contrôle : https://cloudsolutions.vmware.com/content/dam/digitalmarketing/vmware/en/pdf/docs/vmw-sovereign-cloud-technical-guidanc-for-vmware-cloud-provider-partners.pdf

Besoin d'outils pour la mise en œuvre

Il s'agit à la fois de faciliter la migration des applications existantes (avec des outils tel que HCX de VMware), et aussi d'en assurer le contrôle continu (avec https://docs.vmware.com/en/Management-Packs-for-vRealize-Operations/1.0/Compliance-Pack-for-VMware-Sovereign-Cloud/GUID-3DF17387-8399-444F-BD4B-0F2EA9161532.html).

Au-delà de la protection des données existantes, il faut aussi permettre le développement de services innovants. Innover tout en gardant 'le choix et le contrôle'. Ne pas se laisser 'locker' par des PaaS propriétaires. Favoriser l'Open Source pour les composants directement en lien avec le code à développer. C'est ce que propose VMware avec l'offre Tanzu, et plus particulièrement Tanzu Data Services. C'est notamment illustré par l'utilisation de Tanzu Data Services dans Gaia-X (à noter, VMware est Day-1 member de Gaia-X).

Pour aller plus loin dans l'exploration des solutions technologiques VMware met à disposition un environnement dédié :
https://vmc.techzone.vmware.com/vmware-sovereign-cloud
https://www.linkedin.com/groups/12678920/

Database en tant que service : quels sont les choix ?

Développer des solutions sur des bases de données cloud est quelque chose d'assez différent d'un déploiement classique sur un environnement OnPremises. Dans cet article, il est question du Cloud Microsoft Azure, qui est un acteur majeur dans la fourniture de services cloud.



Thierry Bollet

Architecte Cloud Azure pour la société CapGemini.
Il a été nommé Microsoft MVP (Most Valuable Professional) dans la catégorie Microsoft Azure en 2020, 2021 et 2022.
Également auteur aux éditions ENI.
Passionné de Cloud et d'automatisation.

Une base de données dans le Cloud est ce que l'on appelle communément un PAAS, c'est-à-dire une plateforme as a service, même s'il est encore possible de créer des bases sur des machines IAAS (installation d'un socle OS sur une ou des machines virtuelles, puis installation du logiciel de base de données). On ne parle plus dans ce cas de bases « Cloud natives » et ces mises en œuvre sont de plus en plus rares et ne sont pas traitées dans ce sujet. Même réflexion pour les images de la place de marché qui ne sont pas non plus mentionnées dans ce sujet. Il n'est toutefois pas inutile d'en avoir connaissance, puisque la recherche sur le portail affiche… 964 réponses dans la catégorie Databases.

Pas une base, mais des bases : l'exemple d'Azure

Il est donc ici exclusivement question de PAAS. C'est un service managé. À ce titre, il est livré clef en main par le fournisseur de Cloud, sans que le développeur ou le DBA n'aient pas à se soucier de toute la couche d'infrastructure qui présente le service. Je paye une facture pour ce service, et « j'achète » donc de la performance, de la redondance, de la sauvegarde automatique, de l'élasticité / évolutivité, une garantie de disponibilité pour mon service (SLA), etc. La liste n'est pas exhaustive. Dans ces modèles, on « perd » un peu de ce qui fait la finesse de configuration de ses bases, mais on gagne énormément en confort et souplesse d'utilisation. On se concentre sur l'essentiel, l'exploitation de ses données de bases. La répartition des responsabilités change un peu, et le fournisseur de services prend à sa charge tout ou partie de l'infrastructure.

Il peut être intéressant de consulter le tableau comparatif des services de bases de données et de présenter leur domaine de prédilection. Ce tableau est issu de la documentation éditeur et présente en quelques colonnes les principales différences d'une base à l'autre. **Figure 1**

Le choix est vaste même si SQL est très représenté dans ce tableau.

	Azure SQL Database	Azure SQL Managed Instance	SQL Server on Virtual Machines	Azure Database for PostgreSQL	Azure Database for MySQL	Azure Database for MariaDB	Azure Cosmos DB	Azure Cache for Redis
Relational Database	✓	✓	✓	✓	✓	✓		
Non-Relational Database (NoSQL)							✓	
In-Memory Database								✓
Data Models	Relational	Relational	Relational	Relational	Relational	Relational	Multi-Model: Document Wide-column Key-Value Graph	Key-Value
Hybrid	✓	✓	✓	✓ *(Hyperscale)*				
Serverless Compute	✓						✓	
Storage Scale Out	✓ *(Hyperscale)*			✓ *(Hyperscale)*			✓	✓
Compute Scale Out	✓ *(Hyperscale - read-only)*			✓ *(Hyperscale)*			✓	✓
Distributed Multi-Master Writes (Write data to different regions)							✓	✓ *(Coming Soon)*
OSS Based Service (Community edition and open extension support)				✓	✓	✓		✓
HTAP (Available with Azure Synapse Link)	✓ *(Coming Soon)*			✓ *(Coming Soon)*			✓	

Figure 1

Dans la panoplie des services de bases, rien ne manque à l'appel. Des données relationnelles et non relationnelles, mais aussi de la base de données en mémoire avec Azure Cache pour Redis. Alors finalement, quelles différences majeures pour le développeur ?

La première, c'est qu'il peut très facilement s'approprier son environnement et dispose d'une grande autonomie lorsqu'il a besoin de provisionner des ressources. Déployer un environnement ne prend que quelques minutes. Encore faut-il pour que cela soit efficace bien comprendre ce qui guide le choix de tel ou tel service, dans tel ou tel SKU (stock keeping units).

Comme présenté dans le tableau comparatif, Azure, c'est bien évidement toute la famille de services de base de données SQL Server, mais c'est bien plus que ça. Les bases de données Open Source MySQL, MariaDB et PostGreSQL sont de la partie. Ainsi que la base de données NoSQL maison Cosmos DB. Toujours sur le modèle de services managés.

Comme il n'est pas possible de présenter tous les modèles de base de données, un focus est fait dans cet article sur la partie Azure SQL et ses déclinaisons. Mais le sujet peut être abordé de la même manière pour d'autres solutions managées, c'est le même genre de réflexion qui guidera l'ensemble des choix.

Chaque modèle de base de données à son usage

Si le choix de la technologie, du modèle de base de données est guidé par l'application, chaque modèle va ensuite proposer différents niveaux de services. Et c'est un peu là ou tout commence. Un exemple parlant, les différents niveaux de SKU Azure SQL Database. On démarre avec la base Standard qui est facturée 5€ / mois, mais on a également dans la liste de la base critique pour l'entreprise dotée de 128 vCore à plus de 207 000 € / mois. Il y a entre ces deux extrêmes toute une panoplie de possibilités pour le développeur.

Il n'y aura pas d'autres solutions que de se tourner vers la documentation éditeur pour avoir accès aux nombreux tableaux qui guideront le développeur et / ou l'administrateur de la base de données. Mais dans cette liste, il y a à mon sens de vrais marqueurs. Des choix structurants. Voici quelques pistes pour que le choix soit le plus pertinent possible.

Par exemple, quel type de serveur hébergera ma base. J'ai besoin d'un choix équilibré entre la mémoire et le calcul, ou j'ai besoin d'optimisation de la mémoire. Ou même d'une infra de type « Confidential computing » ? La mise en place se fera de manière transparente, mais l'infrastructure sous-jacente ne sera pas la même en terme de type de machines virtuelles. Car les machines virtuelles qui vont supporter le service seront plus ou moins spécialisées. C'est une option à cocher en cours de déploiement.

Mêmes questions en termes de nombres de replicas, d'IOPS disques, de type de provisionnement (mes ressources de calcul sont pré-allouées ou bien elles sont étendues à la volée en cas de besoin ?), de redondance du stockage. Voir même la possibilité de mettre en pause (!!?) ma base si elle n'est plus sollicitée pendant une durée choisie. La liste des options est assez étendue.

Ce qu'il faut retenir de ces différentes informations : c'est que la puissance n'est pas le seul critère de choix. Loin de là. La possibilité d'évoluer au travers d'un même SKU est plutôt ce qui fera le choix final. Les possibilités d'évolution du modèle et ses limites sont-elles compatibles avec ce que je dois obtenir en termes de capacités techniques. Et là, IMPOSSIBLE de ne pas se plonger dans la liste des possibilités offertes par les modèles. Je ne peux pas présenter l'ensemble des options, il existe des tableaux de comparaison très clairs qui exposent l'ensemble des points techniques. Quelques sujets sur lesquels il ne faut pas faire l'impasse et qu'il faut absolument prendre en compte :

- Le type de stockage. SSD en local ou SSD distants ?
- Le nombre d'IOPS.
- Nombre maximal de sessions simultanées.
- Latence d'E/S de lecture et écriture.
- La taille du stockage.
- La version supportée. Par exemple, PostGreSQL 10 et 11. Les versions 9.x ont été retirées.

Je le répète : il existe pour chaque type de base une liste exhaustive des options et des limites dans la documentation en ligne. Un passage obligatoire.

A cet instant, la possibilité de ne payer que ce qui est consommé sans s'engager durablement est un avantage incroyable. Au moins dans la phase maquette / POC, se tromper est … sans impact majeur. Faire et défaire en 3 clics ou en infra as code (ce qui est un très bonne pratique) pour repartir sur d'autres types de solutions.

Attention, post déploiement pour la solution finale, même s'il y a beaucoup de souplesse dans l'utilisation des services PAAS et si certaines options sont modifiables sans impact sur la base de données, il peut y avoir des choix définitifs qui obligeront à repartir de zéro. Rappel utile, tant il est facile de changer son infrastructure. Il y a tout de même quelques limites à l'agilité…

Autre rappel utile, la puissance disponible de façon presque instantanée entraine une possible dérive lors de la conception. Il devient facile de « masquer » ses erreurs de développement en augmentant de manière infinie (ou presque infinie) la puissance des services. Si cela permet de proposer une expérience correcte aux utilisateurs finaux, attention à la facturation. Le payeur va rapidement demander de justifier le volume des dépenses. Il peut y avoir une augmentation tem-

ADO.NET JDBC ODBC PHP Go

PHP (SQL authentication)

```php
<?php
// PHP Data Objects(PDO) Sample Code:
try {
    $conn = new PDO("sqlsrv:server = tcp:srvprogrammez.database.windows.net,1433; Database = magazine-programmez", "adminprogrammez", "{your_password_here}");
    $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
}
catch (PDOException $e) {
    print("Error connecting to SQL Server.");
    die(print_r($e));
}

// SQL Server Extension Sample Code:
$connectionInfo = array("UID" => "adminprogrammez", "pwd" => "{your_password_here}", "Database" => "magazine-programmez", "LoginTimeout" => 30, "Encrypt" => 1, "TrustServerCertificate" => 0);
```

Download PHP driver for SQL server

Figure 2

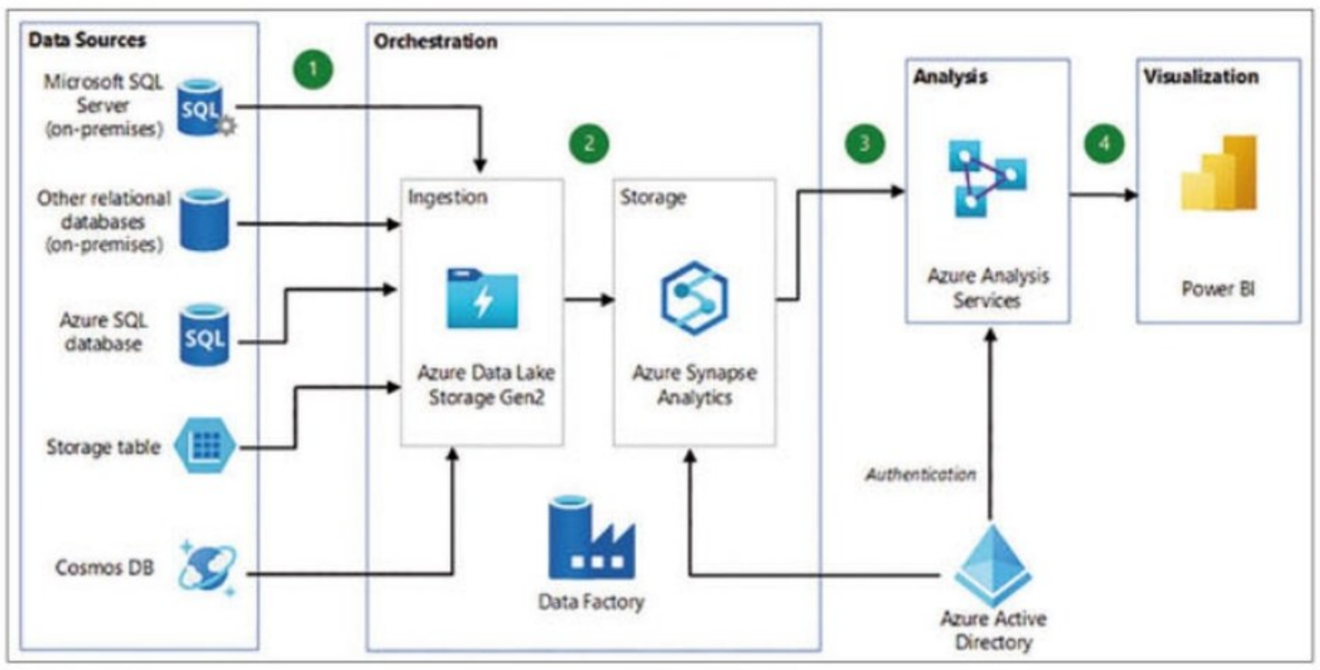


Figure 3

poraire dans certaines phases du cycle de vie de l'application. Pour que les équipes de développement viennent corriger tel ou tel bug ou parce qu'elles ont besoin d'optimiser une partie de l'application. Cet accroissement temporaire permet de ne pas dégrader l'expérience utilisateur tout en donnant « un peu d'air » au développement, mais il ne doit pas être définitif. Il faut avoir ce point en tête et garder un œil sur sa facturation.

Il reste maintenant à traiter les sujets d'exploitation de ces bases. Comment le développeur peut encore et encore améliorer son expérience. En termes d'outils, je pense que le développeur doit avoir accès au portail. PAAS Azure = une plus grande autonomie. Toutes les limites, possibilités et autorisations sont fixées entre les équipes Cloud ó DBA ó Développeur. Équilibre parfois difficile.

Voici quelques points à traiter pour donner / obtenir de l'autonomie. Je parle ici d'exemples concrets, ce que j'ai à mettre en place lorsque j'ai à fournir du PAAS à des équipes de développement.

- Un accès à la base, si possible sur un point de terminaison privé. Par défaut, un service PAAS est exposé publiquement.
- Un accès étendu pour profiter de fonctionnalités avancées comme l'exploitation d'Azure Monitor, l'accès aux métriques des droits sur la restauration des bases, les outils d'analyses ou de sécurité…etc. Ou même la récupération des chaînes de connexions qui sont proposées dans le menu de la base selon le type de langage. **Figure 2**
- L'installation de « Azure Data Studio » pour une gestion centralisée des bases ou d'un outil équivalent pour des bases Open Source comme pgAdmin pour PostGreSQL.
- Une formation avancée aux différentes fonctionnalités comme la restauration d'une sauvegarde appelée PITR (point-in-time restore). Une courte parenthèse sur cette fonctionnalité qui permet au développeur d'un simple clic de souris de restaurer en toute autonomie une base dans un état antérieur sur une autre instance. Il y a ici un point de vigilance particulier. L'expérience montre que cette facilité de restauration doit être accompagnée d'action de nettoyage des bases restaurées. Car les instances provisionnées lors de la restauration sont facturées et viendront impacter fortement les coûts s'il n'y a pas d'actions post restauration.
- Un œil avisé en pré étude sur les modèles de références disponibles à cette adresse : https://learn.microsoft.com/fr-fr/azure/architecture/browse/. Un peu plus de 140 diagrammes et descriptions techniques pour la catégorie « Bases de données ».

Ce dernier lien est indispensable tant les services liés sont variés. Voir à titre d'exemple ci-dessous une architecture complète pour l'entreposage et l'analytique des données. Une architecture hybride On prem / Cloud. **Figure 3**

Comme on le voit, le développeur pourrait avoir à intervenir à différents niveaux dans ce genre d'architecture. Au niveau des Datasources, de l'orchestration, de l'analyse et même de la visualisation.

Avec ces différentes bases de réflexion, il faudra au travers de tests et de POC venir conforter ses choix techniques. C'est une nouvelle ère qui s'est ouverte ces dernières années et une belle extension d'usage pour les métiers du développement ! Quelques changements dans les habitudes de travail, dans le sens d'un accroissement de l'autonomie et d'une souplesse de travail accrue.

Les anciens numéros de PROGRAMMEZ!
Le magazine des dévs



Disponible sur www.programmez.com

Cockpit : la plateforme de gestion des ressources sur le Cloud d'OUTSCALE

Parce que déployer ses ressources cloud devrait être aussi rapide et facile que les gérer, OUTSCALE a développé sa propre interface web de déploiement cloud appelée Cockpit.

Présentation de Cockpit

L'interface cockpit d'OUTSCALE est une interface web qui permet aux utilisateurs des services OUTSCALE de gérer leur infrastructure cloud à partir de son navigateur. Cockpit est une console de gestion d'OUTSCALE permettant de surveiller l'état des ressources cloud, de créer et de configurer des machines virtuelles, des réseaux, et des snapshots, de gérer des groupes de sécurité et des images machines (OMI), et bien plus encore.

L'interface utilisateur de Cockpit est simple et intuitive, ce qui la rend facile à utiliser même pour les débutants. Cockpit permet également aux utilisateurs de visualiser des tableaux de bord en temps réel pour surveiller l'utilisation des ressources et les performances des machines virtuelles. De plus, l'interface cockpit d'OUTSCALE est compatible avec la plupart des navigateurs web et est disponible pour répondre aux besoins des utilisateurs sur les différentes régions Cloud.

Pourquoi utiliser cockpit ?

Conçue pour aider les développeurs à déployer et gérer des applications dans le cloud, l'interface de gestion de ressources apporte des fonctionnalités et une meilleure expérience utilisateur pour les clients qui cherchent à optimiser leur workflow. **Figure 1**

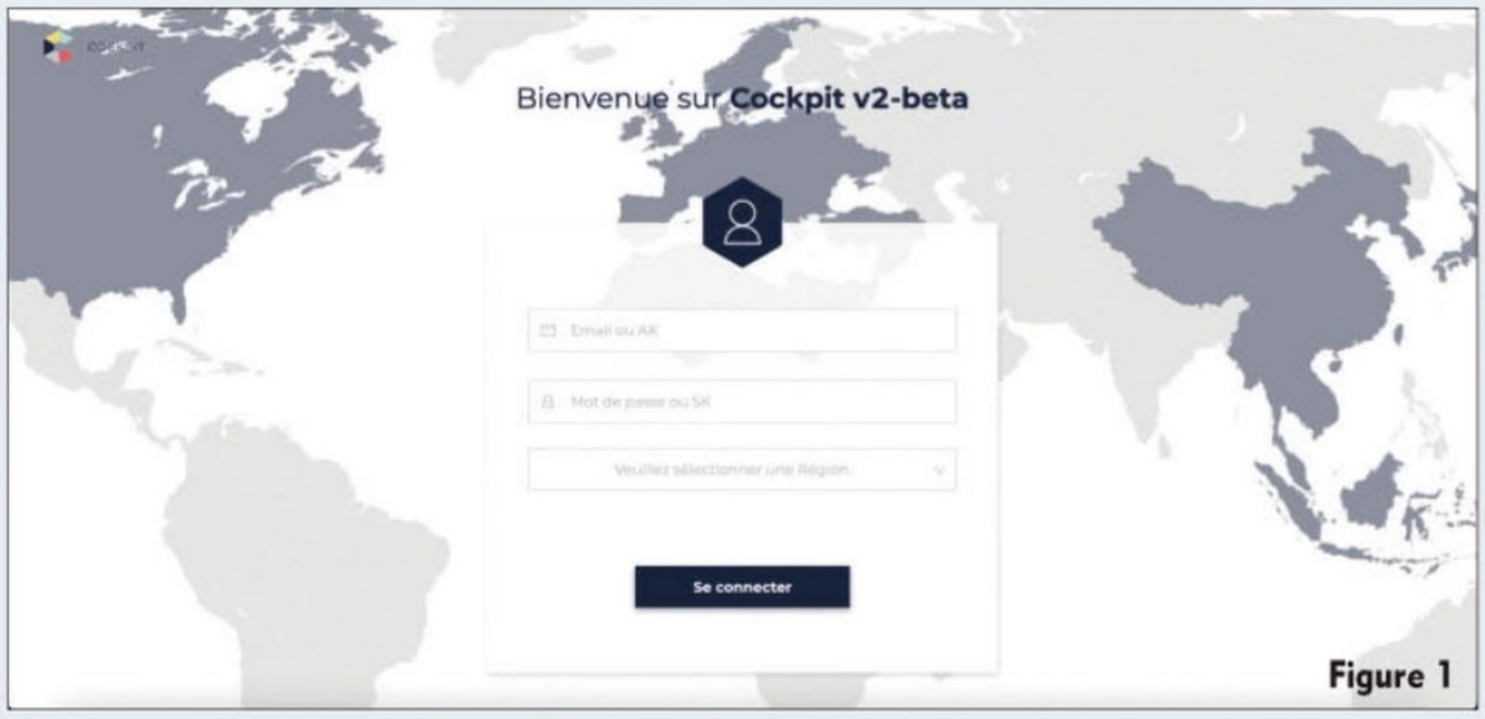


Figure 1

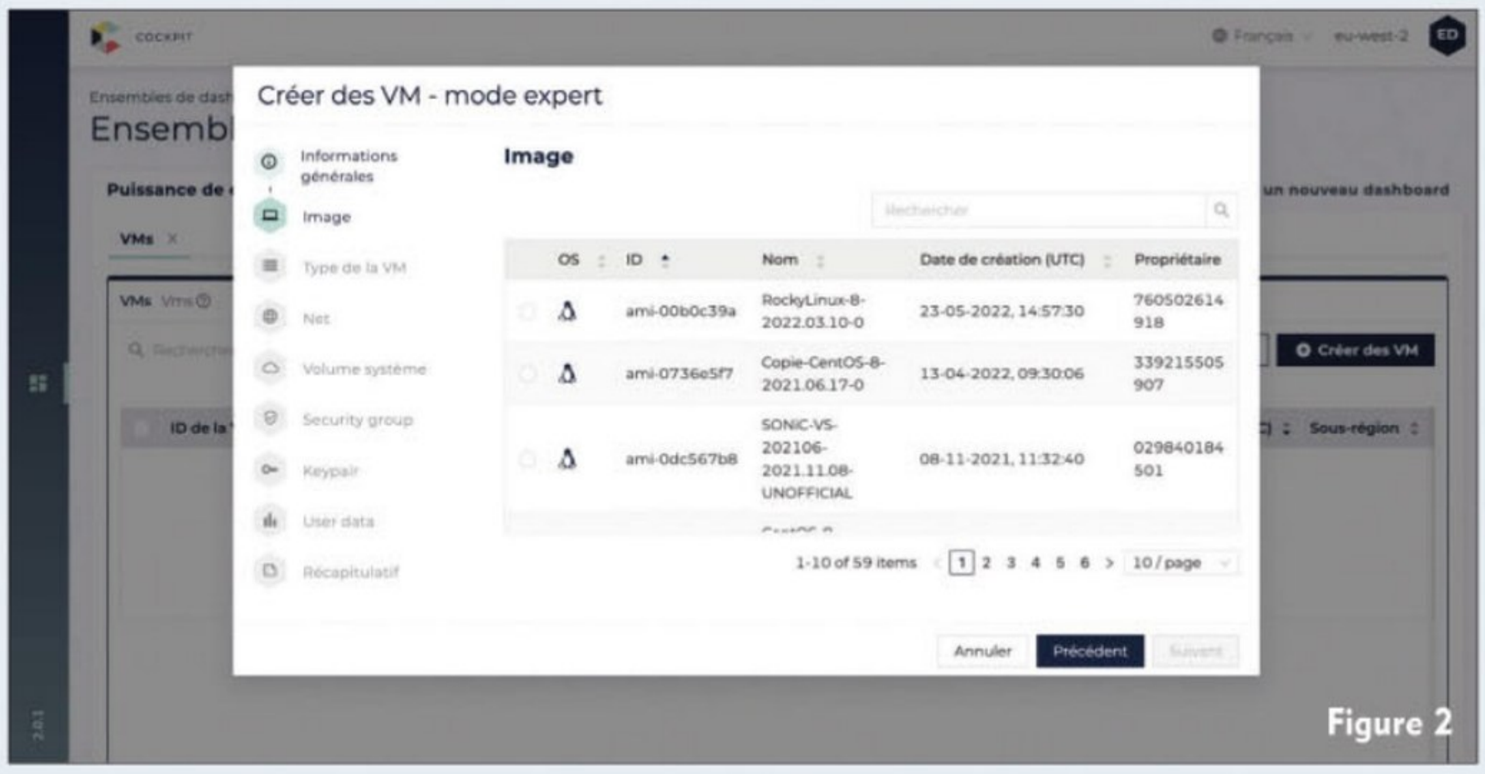


Figure 2

Les utilisateurs choisissent de gérer leur infrastructure cloud pour plusieurs raisons :

- Interface utilisateur intuitive : l'interface est conçue pour être simple et intuitive, ce qui facilite la gestion des ressources cloud pour les utilisateurs, peu importe leur niveau en connaissances techniques.
- Gestion centralisée des ressources : Cockpit permet de gérer toutes vos ressources cloud depuis une interface unique, ce qui est un gain de temps, d'efficacité et de productivité.
- Surveillance des performances : Cockpit vous permet de surveiller en temps réel l'utilisation des ressources et les performances de vos machines virtuelles, ce qui vous permet de prendre des mesures rapidement en cas de problèmes.
- Sécurité renforcée : Cockpit dispose de fonctionnalités de sécurité avancées, telles que la gestion des groupes de sécurité et l'authentification à deux facteurs, qui vous permettent de protéger votre infrastructure cloud contre les attaques malveillantes.
- Gestion multi-cloud : Cockpit est compatible avec plusieurs clouds à partir d'une seule interface.

En résumé, utiliser cockpit d'OUTSCALE peut aider à gérer une infrastructure cloud de manière plus efficace, en offrant une interface utilisateur intuitive, une gestion centralisée des ressources, une surveillance des performances en temps réel et une sécurité renforcée. **Figure 2**

Comment déployer des ressources sur Cockpit ?

Le déploiement de ressources sur cockpit peut être effectué en suivant ces étapes :

1. Connectez-vous à votre compte OUTSCALE : tout d'abord, connectez-vous à votre compte OUTSCALE en accédant à la page de connexion à l'aide de vos identifiants.
2. Accédez à l'interface cockpit : Une fois connecté, accédez à l'interface cockpit en sélectionnant la région où vous souhaitez déployer vos ressources et en cliquant sur le bouton "Cockpit" correspondant.
3. Créez des ressources : dans l'interface cockpit, vous pouvez créer de nouvelles ressources telles que des machines virtuelles, des réseaux, des disques, des images, des groupes de sécurité, etc. Pour créer une nouvelle ressource, cliquez sur le bouton correspondant dans le menu de gauche, puis remplissez le formulaire avec les informations requises.

4 Configurez vos ressources : après la création de vos ressources, vous pouvez les configurer en fonction de vos besoins en utilisant l'interface cockpit. Vous pouvez, par exemple, configurer la taille et le type de machine, ajouter des disques, configurer des règles de sécurité, etc.

5 Lancez vos ressources : une fois que vos ressources sont configurées, vous pouvez les lancer en cliquant sur le bouton "Créer" dans l'interface cockpit. Vos ressources seront alors déployées sur le cloud OUTSCALE dans la région sélectionnée en toute sécurité.



Figure 3 : *Fichier de configuration avec la CLI OUTSCALE*
La ligne "client certificate" indique le chemin vers le certificat.

Découverte de la console de management

L'authentification forte au cœur de la console de management

Pour garantir l'hébergement des ressources dans un environnement cloud de confiance, la sécurité joue un rôle essentiel dans la gestion des ressources des utilisateurs cloud.

Ancrée dans l'ADN d'OUTSCALE et de ses services, l'authentification renforcée est un facteur clé pour assurer la sécurité des ressources manipulées sur le compte cloud de chaque utilisateur.

Pour cela et pour faire ses premiers pas sur le Cloud OUTSCALE, il est essentiel de s'assurer que l'utilisateur qui tente de se connecter soit bien la personne autorisée pour accéder aux ressources.

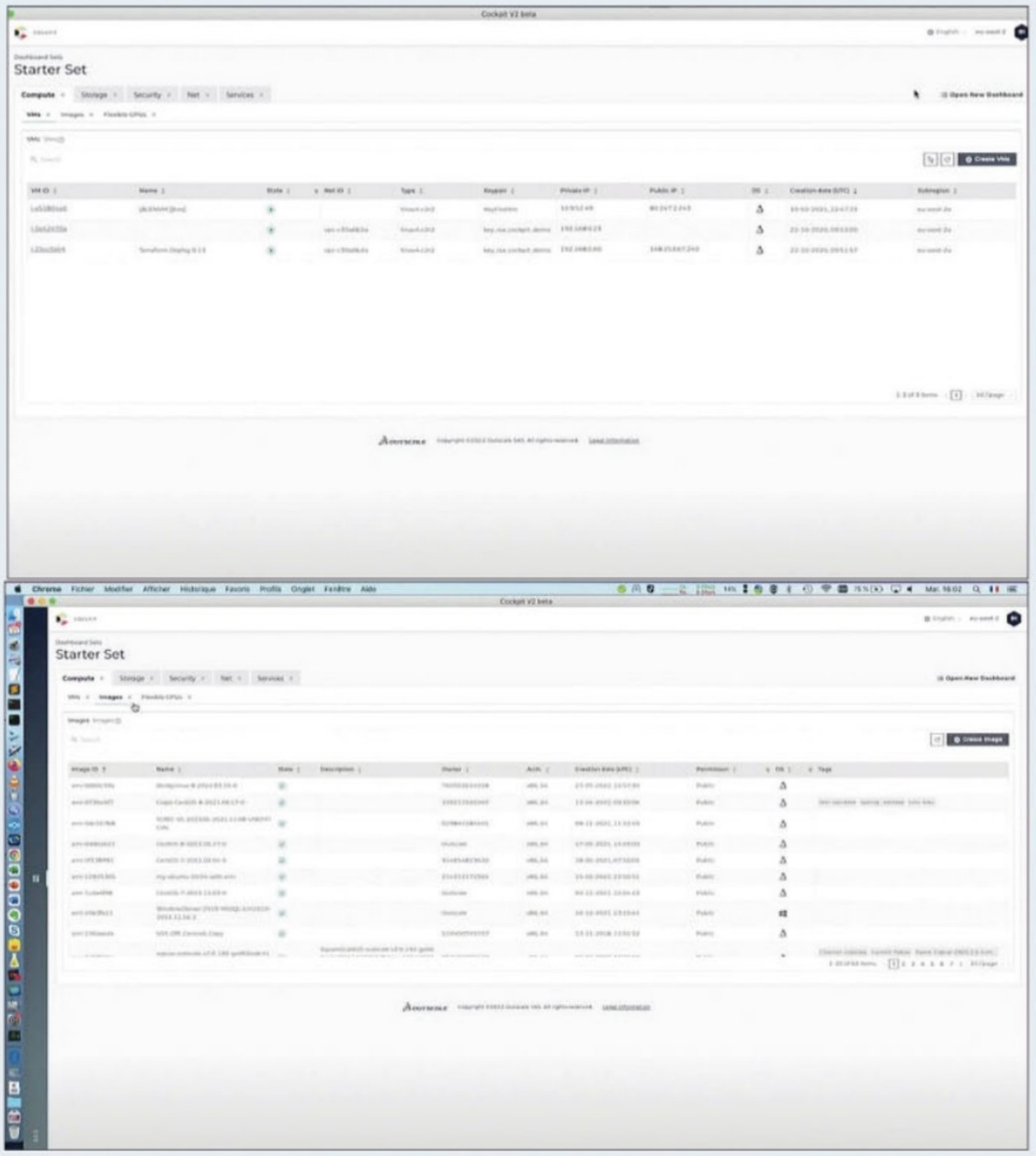
Les différents facteurs d'authentification

Il existe 3 familles de facteurs d'authentification :

- facteur de connaissance (le plus répandu)
- facteur de possession
- facteur d'appartenance

Quand on se connecte sur une application à l'aide d'un simple login et mot de passe, ce sont des éléments connus, c'est donc un **facteur de connaissance**.

Pour aller plus loin dans l'authentification, il y a le **facteur de possession**, c'est-à-dire les preuves qui correspondent à ce que je possède. Par exemple, un téléphone portable sur lequel je vais recevoir un code à usage unique via un SMS. Mais, seul, celui-ci reste un facteur "faible" dans le sens où une personne mal intentionnée peut passer derrière vous, le lire, et donc le subtiliser. Le badge ou la clé USB sécurisée sont également des facteurs de possession. Les certificats peuvent servir de facteur de possession lorsqu'ils sont stockés sur des dispositifs physiques adaptés tels que des cartes à puce.



Figures 4 : *Dashboard des ressources*
Exemple : Dans l'onglet Compute, on retrouve des sous-onglets liés au service compute avec VM, OMI (OUTSCALE Machine Image, GPU)

Le **facteur inhérent ou d'appartenance** est lié à ce que je suis. Exemple : une empreinte vocale, une empreinte digitale ou empreinte rétinienne. Attention ! Celui-ci est irrévocable, on ne peut le changer.

Sur la plateforme Cockpit, OUTSCALE a donc fait le choix de partir sur une double authentification possible avec les facteurs de connaissance et de possession. L'authentification multi-facteur (MFA) est clé dans cette nouvelle console Cockpit, pour

garantir toujours plus de sécurité aux utilisateurs. Pour renforcer la sécurité du compte, il est pertinent de combiner des règles, aussi appelées "API access Rules".

6 étapes pour activer l'authentification multi-facteurs sur votre compte cloud :

1. Générer une Autorité de Certification (CA) auprès de notre orchestrateur TINA OS
2. importer la CA dans le Cloud
3. Configurez vos règles d'accès API
4. Testez votre accès aux API
5. Testez votre certificat client avec une Command Line Interface (CLI) ou Terraform
6. Activez une session de confiance pour améliorer l'expérience utilisateur (pas obligatoire)

La session de confiance permet de stocker le certificat, d'avoir une session qui va permettre d'exécuter des calls API sans présenter de façon systématique le certificat.

En résumé : En phase avec les recommandations de l'Agence nationale de la sécurité des systèmes d'information (ANSSI), le MFA chez OUTSCALE c'est : un facteur de connaissance et un certificat x 509 sur un support physique, qui devient un facteur d'appartenance.

Access Key avec OSC CLI **Figure 3**

API Access Rules

L'élément clé d'authentification de cette console Cockpit est l'API Access Rules (AAR). Ce sont les règles que l'utilisateur va pouvoir déclarer auprès de TINA OS (orchestrateur cloud d'OUTSCALE). L'API Access Rules est l'élément où seront rassemblés les différents critères qui vont autoriser ou non les accès des utilisateurs du Cloud.

On y retrouve : une adresse IP source que l'utilisateur souhaite autoriser, l'ID du certificat. On peut donc avoir plusieurs critères par API Access Rules.

Il va falloir donc combiner "ce que je connais" (facteur de connaissance) et "ce que je possède" (facteur de possession).

À noter : si on active la session de confiance, les appels API qui peuvent être utilisés dans le cadre de la session de confiance sont des calls pour déployer des machines virtuelles, déployer du stockage, des volumes... Si l'utilisateur souhaite modifier la stratégie d'accès c'est-à-dire s'il souhaite modifier une API Access Rules, un certificat x 509 sera demandé parce qu'il vient de modifier un élément permettant de vérifier et de prouver l'identité d'un utilisateur.
La session de confiance, se concrétise par l'API access Policy.

Pour activer le Trusted Environment (TE) ou une Session de confiance, l'utilisateur doit :

- Vérifier L'ensemble des Api Access Rules (AAR) du compte doivent utiliser une vérification par certificat.
- L'utilisateur doit résilier l'activation du TE (TrustedEnv = True) avec un Call authentifié avec un certificat.
- Le paramètre MaxAccessKeyExpirationSeconds doit être initialisé avec une valeur finie.
- L'ensemble des AK/SK du compte doivent avoir une date d'expiration finie ne dépassant pas la valeur maximum renseignée dans le paramètre MaxAccessKeyExpirationSeconds.

Une fois la session de confiance initialisée, pour interagir avec le Cloud, le certificat devient facultatif.

Une fois authentifié dans la console, différents onglets sont proposés par service disponible : calcul, stockage, sécurité/réseau...) **Figures 4**

Pour profiter pleinement de l'expérience cloud OUTSCALE, et de son interface Cockpit, scannez le QR code :

Créé en 2010, OUTSCALE, marque de Dassault Systèmes, est un opérateur souverain et durable de business experience de confiance en mode service. Sa mission est de proposer une architecture unique de services, qui comprend un large éventail d'offres adaptées pour la création d'expérience de jumeaux virtuels. OUTSCALE offre aux organisations publiques et privées des expériences métiers accessibles en continu, de manière simple, sécurisée et durable.
Présent dans 3 régions dans le monde (Europe, Amérique, Asie), OUTSCALE propose 3 modèles de Cloud : souverain, international et dédié.

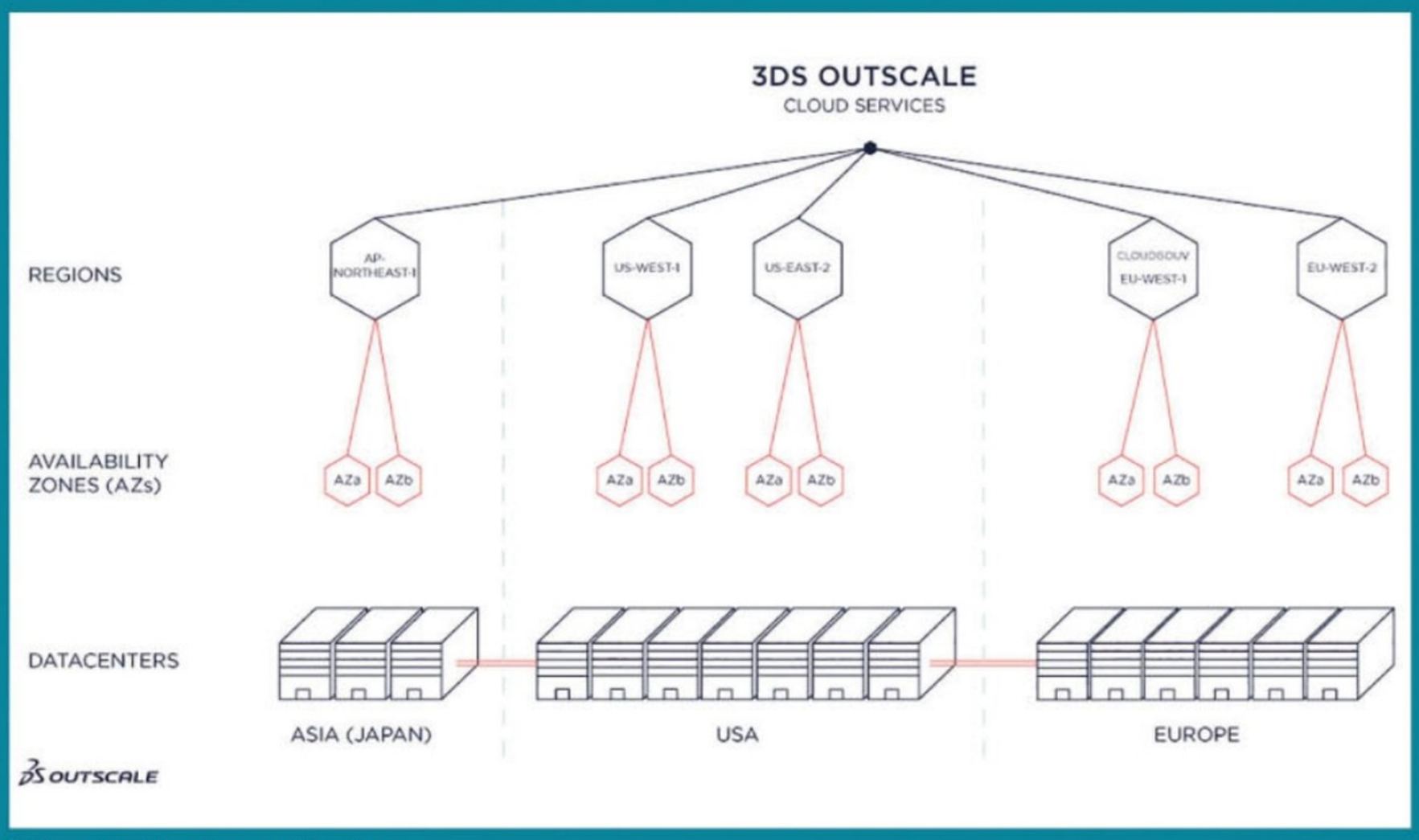


Schéma des différentes zones disponibles

Gérer facilement son infrastructure avec le provider Terraform OVHcloud

Terraform est devenu de facto l'outil de référence pour faire de l'Infrastructure as Code. Mais saviez-vous que vous pouvez utiliser votre outil préféré pour monter votre infra chez OVHcloud ?



Aurélie Vache

DevRel chez OVHcloud

Google Developer Expert

Docker Captain

CNCF Ambassador

GitPod Hero

Aurélie est speakeuse, sketchnoteuse et rédactrice d'articles et de livres techniques.

Organisatrice de conférences.

Créatrice et auteur de "Understanding Kubernetes in a visual way" et de "Understanding Docker in a visual way" en sketchnotes, vidéos et livres (livres disponible sur Amazon & GumRoad)

@aurelievache

OVHcloud







Figure 1

Terraform est un outil d'Infrastructure as Code (IaC) open source créé par Hashicorp en 2014, écrit en Go. Il permet de **construire**, **modifier** et **versionner** une **infrastructure immuable**. **Figure 1**

Immuable signifie que vous pouvez exécuter plusieurs fois vos fichiers Terraform, et le résultat sera toujours le même.

L'outil dispose d'une interface en ligne de commande (CLI) *terraform* puissante et très intuitive.

Si cela vous intéresse, j'ai créé une cheat sheet afin de lister les commandes les plus utiles avec des exemples concrets : https://github.com/scraly/terraform-cheat-sheet/blob/master/terraform-cheat-sheet.pdf

On peut définir et provisionner son infrastructure en écrivant la définition de nos ressources en Hashicorp Configuration Language (HCL) dans un fichier se terminant par **.tf**. **Figure 2**

Terraform est modulable et extensible via une architecture de providers.

Depuis son fichier de configuration, on peut appeler plusieurs providers, qui appellent la library cliente et enfin l'API distante. **Figure 3**

Il existe un nombre impressionnant de providers publiés dans le registry officiel (https://registry.terraform.io/), plus de 2800 à l'heure où j'écris ces lignes !

Dans ce registry vous pourrez ainsi retrouver les providers pour AWS, GCP, Consul, Atlas, Chef, Kubernetes, OpenStack, … et également OVHcloud ^ ^.

OVHcloud Terraform provider **Figure 4**

Tout comme GCP (Google Cloud Platform) et AWS (Amazon Web Services), OVHcloud dispose d'un provider Terraform afin d'interagir avec un large éventail de produits et de services.

Le provider Terraform OVHcloud est disponible sur le registry officiel : https://registry.terraform.io/providers/ovh/ovh/latest

A l'heure ou j'écrit ces lignes, la version 0.27.0 vient de sortir : **Figure 5**

Le provider Terraform, open source, est maintenu par des équipes internes chez OVHcloud mais également par des contributeurs externes. **Figure 6**



Figure 2



Figure 3

Presque une centaine de *resources* et de *datasources* sont à disposition actuellement et "ce n'est pas fini" ^ ^, de nouveaux produits sont intégrés à chaque nouvelle release :).
Le provider supporte actuellement :

- Bare Metal Cloud (serveurs dédiés, VPS, NAS-HA,...
- LDP (Logs data Platform)
- Managed Databases (ainsi que leurs utilisateurs, ACL, IP restrictions ...)
- S3 (utilisateurs ...)
- Network (private network, subnet, vRack, Public IPs...)
- Load Balancer
- Managed Kubernetes (ainsi que les nodes pools, IP restrictions, OIDC...)
- Managed Private Registry (ainsi que les utilisateurs ...)
- Public Cloud (projet, utilisateurs, clés SSH ...)
- Web Cloud (nom de domaines, zones DNS, ...)
- Web hosting Databases
- Dedicated servers
- ...

Toutes les resources et datasources disponibles sont documentées : https://registry.terraform.io/providers/ovh/ovh/latest/docs

Comment utiliser le provider OVHcloud ?

Trève de blabla, passons à la partie pratique de cet article.
Pour utiliser le provider OVHcloud, la première étape est de spécifier que l'on veut utiliser le provider "ovh" qui se trouve dans le namespace "ovh" dans le registry officiel de Terraform (*source = "ovh/ovh"*). Ensuite on doit spécifier quelques attributs pour faire fonctionner le provider : un endpoint ainsi que ses credentials pour utiliser l'API.
Créez un fichier **provider.tf** :
Pour Terraform 0.13 ou supérieur :

```
terraform {
 required_providers {
  ovh = {
   source = "ovh/ovh"
  }
 }
}

provider "ovh" {
 endpoint        = "ovh-eu"
 application_key   = "<your_access_key>"
 application_secret = "<your_application_secret>"
 consumer_key     = "<your_consumer_key>"
}
```

Pour Terraform 0.12 ou inférieur :

```
# Configure the OVHcloud Provider
provider "ovh" {
 endpoint        = "ovh-eu"
 application_key   = "<your_access_key>"
 application_secret = "<your_application_secret>"
 consumer_key     = "<your_consumer_key>"
}
```

Vous pouvez, de manière alternative, définir les credentials dans des variables d'environnement.

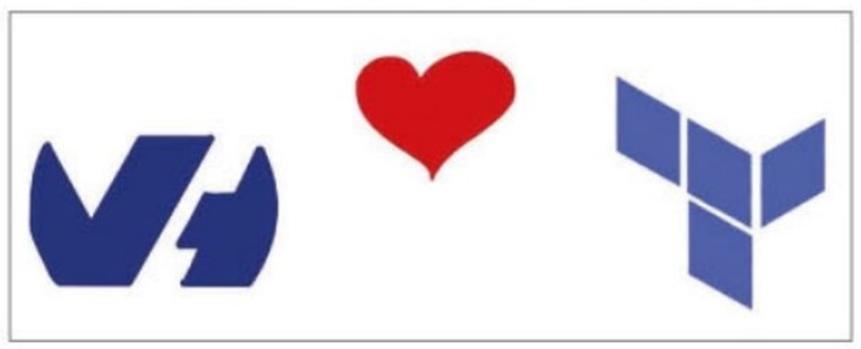
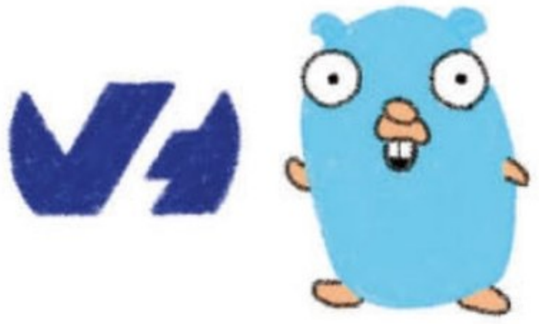
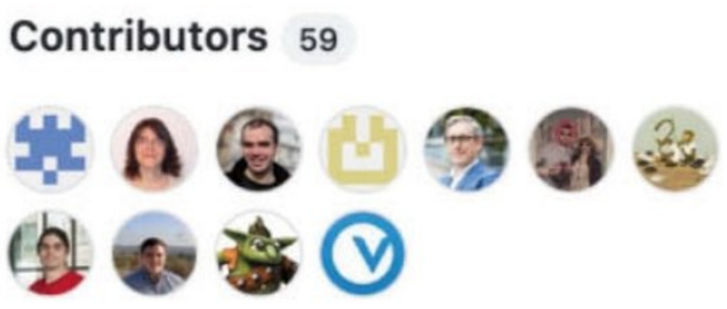
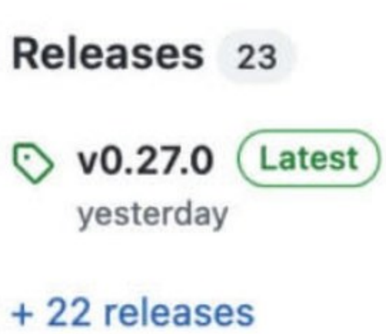


Figure 4

Figure 6







Contributors 59

+ 48 contributors

Releases 23

v0.27.0 Latest
yesterday

+ 22 releases

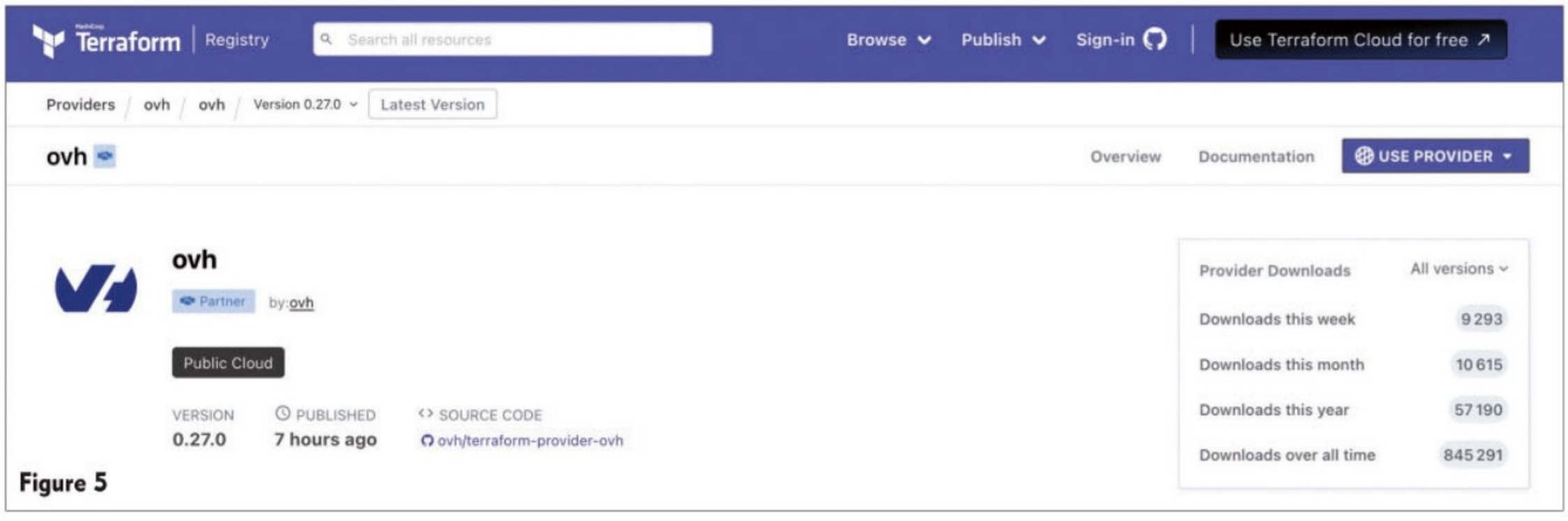


Figure 5

- OVH_ENDPOINT
- OVH_APPLICATION_KEY
- OVH_APPLICATION_SECRET
- OVH_CONSUMER_KEY

Cette dernière méthode (ou une alternative similaire) est recommandée pour éviter de stocker des données secrètes dans un repository.
Afin de générer les credentials, merci de consulter la documentation :
https://registry.terraform.io/providers/ovh/ovh/latest/docs#provider-configuration

Dans cet article, nous avons défini le endpoint **ovh-eu** car nous voulons appeler l'API OVHcloud Europe, mais d'autres endpoints existent, selon vos besoins :

- ovh-eu pour l'API OVHcloud Europe
- ovh-us pour l'API US d'OVHcloud
- ovh-ca pour l'API OVHcloud Amérique du Nord

A cet instant le provider est utilisable et nous pouvons demander à Terraform d'initialiser le projet :

```
$ terraform init

Initializing the backend...

Initializing provider plugins...
- Finding latest version of ovh/ovh...
- Installing ovh/ovh v0.27.0...
- Installed ovh/ovh v0.27.0 (signed by a HashiCorp partner, key ID F56D
1A6CBDAAADA5)

Partner and community providers are signed by their developers.
If you'd like to know more about provider signing, you can read about it here:
https://www.terraform.io/docs/cli/plugins/signing.html

Terraform has created a lock file .terraform.lock.hcl to record the provider
selections it made above. Include this file in your version control repository
so that Terraform can guarantee to make the same selections by default when
you run "terraform init" in the future.

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan"
 to see any changes that are required for your infrastructure. All Terraform
 commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
```

La commande **init** initialise votre répertoire de travail qui contient les fichiers de configuration **.tf**.
C'est la première commande à exécuter pour une nouvelle configuration, ou après avoir cloné un nouveau repository git ayant des fichiers **.tf** à l'intérieur.

La commande **init** va :

- Télécharger et installer les providers Terraform
- Initialiser le backend (si défini)
- Télécharger et installer des modules (si défini)

Maintenant nous allons, par exemple, créer un cluster Kubernetes.
Pour cela, créez un fichier variables.tf avec votre **service_name**:

```
variable service_name {
 type      = string
 default   = "<your_service_name>"
}
```

Note: *La valeur de l'attribut* service_name *est l'ID de votre projet Public Cloud.*

Vous pouvez, de manière alternative, définir le **service_name** dans une variable d'environnement OVH_CLOUD_PROJECT_SERVICE. Par contre, attention, sa valeur n'est pas la même si vous interagissez avec une ressource du Public Cloud ou non. Par exemple, le service_name d'un vRack sera égal a son ID et non à votre ID de projet Public Cloud ;-).
Définissez les ressources que vous souhaitez créer dans un nouveau fichier nommé **ovh_kube_cluster.tf** :

```
resource "ovh_cloud_project_kube" "cluster" {
 service_name = var.service_name
 name         = "my-super-cluster"
 region       = "GRA5"
 version      = "1.25"
}
```

Dans cette définition de ressources, nous demandons à Terraform de créer un cluster Kubernetes, dans la région GRA5, en utilisant la version 1.25 de Kubernetes (la dernière version recommandée au moment où nous avons écrit ce tutoriel). Maintenant, il ne nous reste plus qu'à générer le plan et l'appliquer :

```
$ terraform apply

Terraform used the selected providers to generate the following execution
 plan. Resource actions are indicated with the following symbols:
 + create

Terraform will perform the following actions:

 # ovh_cloud_project_kube.cluster will be created
 + resource "ovh_cloud_project_kube" "cluster" {
    + control_plane_is_up_to_date = (known after apply)
    + id                          = (known after apply)
    + is_up_to_date               = (known after apply)
    + kubeconfig                  = (sensitive value)
    + name                        = "my-super-cluster"
    + next_upgrade_versions       = (known after apply)
    + nodes_url                   = (known after apply)
    + region                      = "GRA5"
    + service_name                = "xxxxxxxxxxxxxxxxxxx"
    + status                      = (known after apply)
    + update_policy               = "ALWAYS_UPDATE"
    + url                         = (known after apply)
    + version                     = "1.25"
 }
```

```
Plan: 1 to add, 0 to change, 0 to destroy.

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes

ovh_cloud_project_kube.cluster: Creating...
ovh_cloud_project_kube.cluster: Still creating... [10s elapsed]
ovh_cloud_project_kube.cluster: Still creating... [20s elapsed]
...
ovh_cloud_project_kube.cluster: Still creating... [3m30s elapsed]
ovh_cloud_project_kube.cluster: Still creating... [3m40s elapsed]
ovh_cloud_project_kube.cluster: Creation complete after 3m47s [id=
76db2764-58d2-4384-b17f-ab38b0c7fc78]

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
```

A noter qu'il existe la commande **terraform plan** qui permet de générer uniquement le plan, mais si nous ne sauvegardons pas le plan dans un fichier de sortie et que nous n'avons pas vocation à l'utiliser (en tant qu'artefact dans un pipeline de CI/CD par exemple), alors la bonne pratique est d'exécuter la commande **terraform apply** directement qui génère le plan et vous permet de l'appliquer en tapant la réponse "yes".
Si vous souhaitez supprimer le cluster Kubernetes que vous avez créé via Terraform, il vous suffit d'exécuter la commande **terraform destroy** :

```
$ terraform destroy
ovh_cloud_project_kube.cluster: Refreshing state... [id=xxxxxx-xxxx-
xxxx-xxxx-xxxxxxxxx]

Terraform used the selected providers to generate the following execution
plan. Resource actions are indicated with the following symbols:
  - destroy

Terraform will perform the following actions:

  # ovh_cloud_project_kube.cluster will be destroyed
  - resource "ovh_cloud_project_kube" "cluster" {
    - control_plane_is_up_to_date = true -> null
    - id                          = "xxxxxx-xxxx-xxxx-xxxx-xxxxxxxxx" -> null
    - is_up_to_date               = true -> null
    - kubeconfig                  = (sensitive value)
    - name                        = "my-super-cluster" -> null
    - next_upgrade_versions       = [] -> null
    - nodes_url                   = "xxxxxx.nodes.xx.gra.k8s.ovh.net" -> null
    - region                      = "GRA5" -> null
    - service_name                = "xxxxxxxxxxxxxxxx" -> null
    - status                      = "READY" -> null
    - update_policy               = "ALWAYS_UPDATE" -> null
    - url                         = "xxxxxx.xx.gra.k8s.ovh.net" -> null
    - version                     = "1.25" -> null
  }

Plan: 0 to add, 0 to change, 1 to destroy.

Do you really want to destroy all resources?
  Terraform will destroy all your managed infrastructure, as shown above.
  There is no undo. Only 'yes' will be accepted to confirm.

  Enter a value: yes

ovh_cloud_project_kube.cluster: Destroying... [id=xxxxxx-xxxx-xxxx-
xxxx-xxxxxxxxx]
ovh_cloud_project_kube.cluster: Still destroying... [id=xxxxxx-xxxx-
xxxx-xxxx-xxxxxxxxx, 10s elapsed]
ovh_cloud_project_kube.cluster: Still destroying... [id=xxxxxx-xxxx-
xxxx-xxxx-xxxxxxxxx, 20s elapsed]
ovh_cloud_project_kube.cluster: Still destroying... [id=xxxxxx-xxxx-
xxxx-xxxx-xxxxxxxxx, 30s elapsed]
ovh_cloud_project_kube.cluster: Destruction complete after 37s

Destroy complete! Resources: 1 destroyed.
```

Conclusion

Comme nous l'avons vu, tout comme avec les autres Cloud providers, il est possible d'utiliser Terraform pour gérer ses services managés et son infrastructure chez OVHcloud.
Le provider est open source et disponible sur GitHub (https://github.com/ovh/terraform-provider-ovh), n'hésitez pas à y contribuer !



Infrastructure As Code avec OVHcloud

PARTIE 1



Olivier Beautier
DevRel chez OVHcloud
Gitpod Community Hero
Comédien improvisateur
@raclepoulpe

OVHcloud

Aujourd'hui, de plus en plus d'entreprises et de particuliers utilisent les services Cloud. La nature de ceux-ci est très variée, et peut dépendre du fournisseur de services choisi. Chez OVHcloud, la gamme est assez large et couvre la majorité des besoins actuels : Bare Metal & VPS (Virtual Private Server), Hosted Private Cloud, Hébergement web & Domaines, Public Cloud, Télécom, etc.

Cependant, ce qui nous différencie, c'est notre volonté de fournir les infrastructures de manière la plus ouverte et transparente possible, sans surcouche ou ajout propriétaire.
Le résultat est que vous avez à disposition une gamme de produits complète, que vous n'avez plus qu'à assembler pour créer votre infrastructure.
Mais comment intégrer au mieux nos produits dans votre environnement, votre processus d'intégration et de déploiements continus ? Comment automatiser, tester et mettre à disposition ces infrastructures aux développeuses et développeurs ?
La réponse est apportée par les outils d'IaC (Infrastructure as Code), tels que Terraform et Ansible pour les principaux, mais la liste est longue.
Que votre entreprise soit organisée en Agile, DevOps ou autre, l'IaC permet de donner plus d'autonomie aux devs (principe du Shift Left) et de facto accélérer vos délais de livraison (Time to Market).
Notre objectif est donc de vous fournir un maximum de moyens pour atteindre les vôtres.

Projet Github d'exemples d'utilisation de l'offre Public Cloud

Afin d'aider les personnes à faire de l'IaC avec les produits Public Cloud de OVHcloud, nous avons créé un projet hébergé chez Github, qui liste des exemples concrêt d'infrastructure à créer avec Terraform : https://github.com/ovh/public-cloud-examples.
Ce projet est né après deux événements marquants chez OVHcloud : tout d'abord l'actualisation de notre **provider Terraform**, sous l'impulsion d'Aurélie Vache, qui nous a permis de couvrir rapidement une grande partie de notre offre.
Ensuite la mise à disposition des **routeurs L3** et **Floating IPs** par nos équipes réseau sur l'offre Public Cloud.
Ce deuxième point ouvre la possibilité de générer à la demande des réseaux et sous-réseaux privés, mais surtout accompagné d'un routeur faisant office de passerelle (gateway), évitant ainsi la génération d'une instance dédiée à cette tâche.
Les Floating IPs, quant à elles, vont permettre de simplifier grandement l'accès à vos composants déployés dans ces environnements privés.
Les avantages de travailler dans un réseau privé sont nombreux. Vous protégez ainsi vos infrastructures des attaques malveillantes, vous pouvez réduire le nombre d'adresses IPv4 publiques, vous maîtrisez votre environnement, etc..
Il n'en fallait pas plus pour commencer à concevoir, sous forme de fichiers de configuration Terraform, des modèles d'infrastructure sécurisés et cohérents pour des applications en environnement de production.

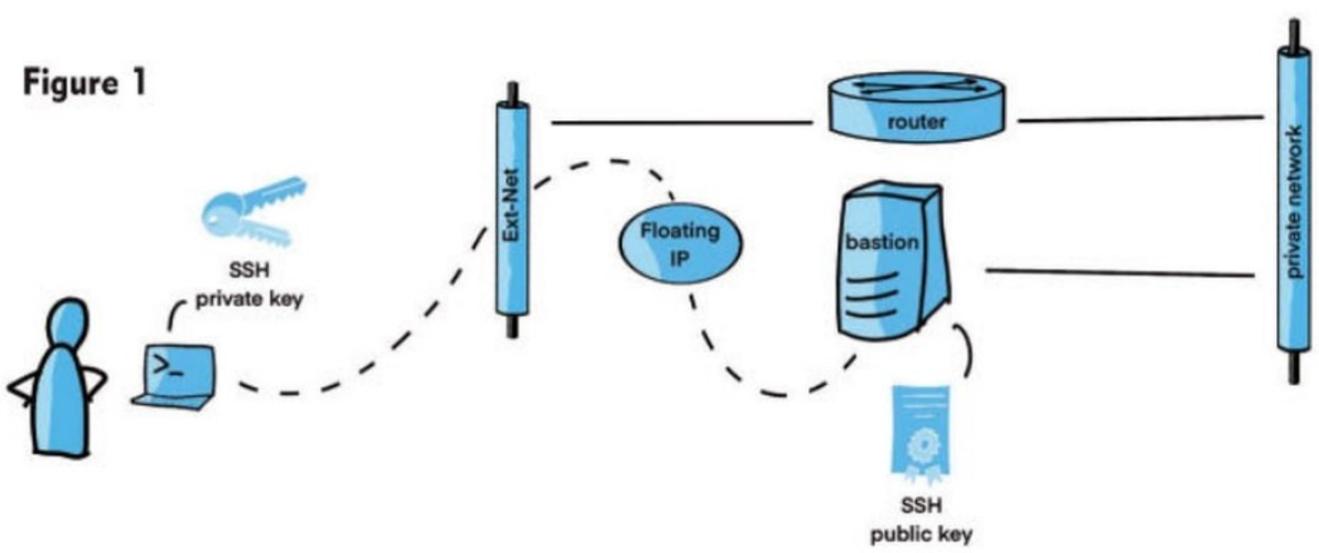
Note : *Il vous faudra quelques pré-requis pour appliquer ces exemples : un compte OVHcloud actif, un projet Public Cloud, les CLI terraform, ansible et un client Git.*

Un premier exemple comme modèle de base : Création d'une instance dans un réseau privé

Figure 1

Dans un premier exemple, nous proposons une architecture basique, pouvant servir de base pour d'autres modèles, déployée sur notre offre Public Cloud.
Il s'agit d'une instance virtuelle, sur laquelle sera déployée une clé SSH, connectée à un réseau privé et accessible via une adresse IP flottante (Floating IP).
Retrouvez en détail cet exemple sur notre projet Github : https://github.com/ovh/public-cloud-examples/tree/main/examples/01
Nous allons l'étudier en détail, afin d'essayer de couvrir un maximum de concepts.
Quand vous commencez à travailler sur un projet Public Cloud (PCI), en coulisse vous travaillez sur un tenant OpenStack. Ceci a pour avantage de vous permettre d'utiliser le provider terraform OpenStack. Cependant, certains composants réseau nécessitent l'emploi du provider OVHcloud, comme la gestion du **vRack**, mais ce n'est pas le cas pour l'instant.
Le point de départ de cet exemple est le fichier *main.tf*. Il fait référence à quatre modules, *private_network*, *ssh_keypair*, *instance_simple* et *floatip*.

Figure 1



Note : *Ces modules ont été créés uniquement dans le but de servir les exemples de ce projet, ils ne sont pas exhaustifs des multiples possibilités de paramétrage de chaque ressources utilisées.*

Chacun de ces modules a en paramètre *source* le chemin vers son bloc de fichiers descriptifs, et les variables nécessaires à son fonctionnement.

```
module "network" {
 source = "../../modules/private_network"
 region = var.region
 network = var.network
 subnet = var.subnet
 router = var.router
}

module "keypair" {
 source = "../../modules/ssh_keypair"
 keypair = var.keypair
}

module "bastion" {
 source   = "../../modules/instance_simple"
 depends_on = [module.keypair, module.network]
 instance  = var.bastion
}

module "floatip" {
 source   = "../../modules/floating_ip"
 depends_on = [module.bastion]
 floatip = {
  region     = var.region
  component_id = module.bastion.instance_id
 }
}
```

Nous retrouvons la définition des variables utilisées par les modules dans le fichier *variables.tf*, et un exemple de valorisation de ces variables dans le fichier *variables.tfvars*.
Par exemple, la variable *subnet*, utilisée par le module *private_network*, sera définie tel que :

```
variable "subnet" {
 description = "Subnet parameters"
 type = object({
  name     = string
  cidr     = string
  dhcp_start = string
  dhcp_end  = string
 })
}
```

et valorisée tel que :

```
subnet = {
 name     = "mySubnet"
 cidr     = "192.168.12.0/24"
 dhcp_start = "192.168.12.100"
 dhcp_end  = "192.168.12.254"
}
```

Ici nous définissons une plage d'adresse DHCP dans le sous-réseau *mySubnet* dont l'adressage CIDR est *192.168.12.0/24*.

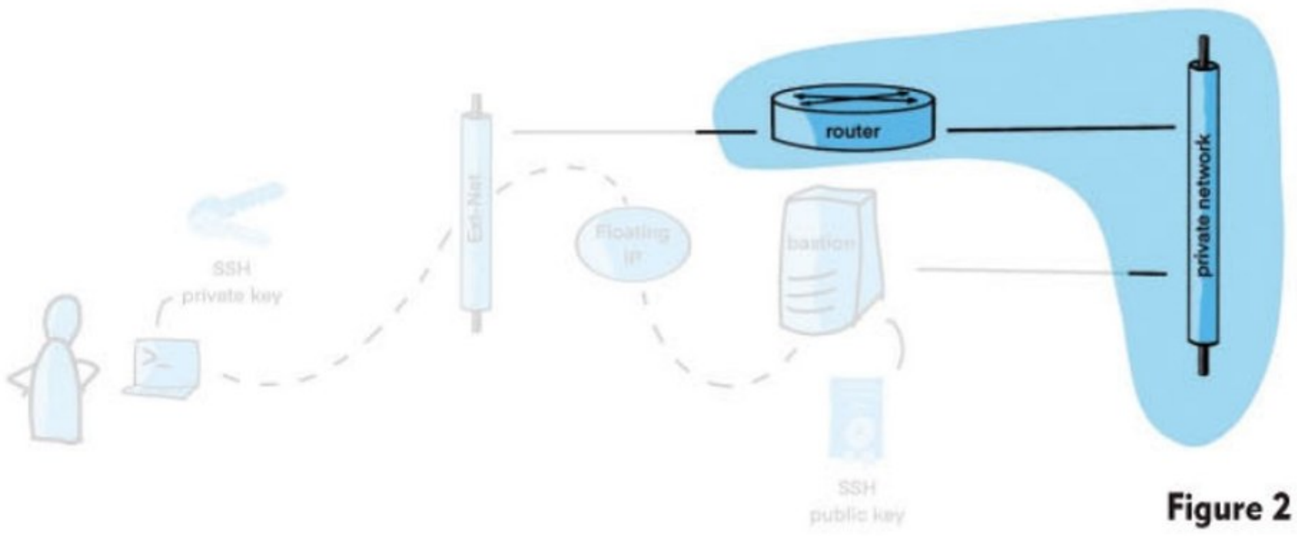


Figure 2

Restons sur ces aspects réseaux pour détailler la première étape de cet exemple.

Réseau privé, sous-réseau et routeur **Figure 2**

La première étape est de créer le réseau privé (private network), sur lequel on crée le sous-réseau (subnet).
C'est donc au module *private_network* que l'ont fait appel.
L'ensemble des ressources nécessaires à ce module est fourni par le provider OpenStack.
Ce provider est configuré au niveau du module, et non au niveau de l'exemple.
Ainsi, tous les modules utilisant OpenStack auront la même configuration.

Configuration du provider OpenStack

La configuration du provider est décrite dans le fichier *providers.tf* du module.
Ici on requiert une version supérieure ou égale à 1.49.0 du provider officiel OpenStack, et une version supérieure ou égale à 1.3.6 pour Terraform :

```
terraform {
 required_providers {
  openstack = {
   source = "terraform-provider-openstack/openstack"
   version = "~> 1.49.0"
  }
 }
 required_version = "~> 1.3.6"
}
```

Le paramétrage de l'accès à votre environnement OpenStack peut se faire au même endroit :

```
# Configure the OpenStack Provider
provider "openstack" {
 user_name  = "admin"
 tenant_name = "admin"
 password  = "pwd"
 auth_url  = "http://myauthurl:5000/v2.0"
 region    = "RegionOne"
}
```

Mais ça n'est pas le cas dans notre exemple. En effet, si comme nous vous ne souhaitez pas faire apparaître vos éléments d'identification dans vos fichiers, il existe d'autres méthodes pour indiquer ces informations au provider (à retrouver sur la page du registre Terraform officielle).
Cet exemple et les autres du projet GitHub préfèrent le positionnement de variables Openstack via un fichier script :

```
#### Openstack vars from openrc file
export OS_AUTH_URL=https://auth.cloud.ovh.net/v3
export OS_IDENTITY_API_VERSION=3
export OS_USER_DOMAIN_NAME=${OS_USER_DOMAIN_NAME:-"Default"}
export OS_PROJECT_DOMAIN_NAME=${OS_PROJECT_DOMAIN_NAME:-"Default"}
export OS_TENANT_ID=""
export OS_TENANT_NAME=""
export OS_USERNAME=""
export OS_PASSWORD=""
export OS_REGION_NAME=""
if [ -z "$OS_REGION_NAME" ]; then unset OS_REGION_NAME; fi
```

Revenons à notre exemple, et à l'utilisation du module private_network, appelé dans le fichier main.tf via

```
module "network" {
 source = "../../modules/private_network"
 region  = var.region
 network = var.network
 subnet  = var.subnet
 router  = var.router
}
```

Pour le faire fonctionner, vous avez à positionner les quatres variables region, network, subnet et router. La structure de ces variables se trouve toujours dans le fichier variables.tf :

```
# Region
variable "region" {
 description = "Region"
 type        = string
}

# Network - Private Network
variable "network" {
 description = "Private Network Parameters"
 type = object({
  name = string
 })
}

# Network - Subnet
variable "subnet" {
 description = "Subnet parameters"
 type = object({
  name       = string
  cidr       = string
  dhcp_start = string
  dhcp_end   = string
 })
}

# Network - Router
variable "router" {
 description = "Router Parameters"
 type = object({
  name = string
 })
}
```

Pour instancier les ressources avec vos valeurs, modifier celles définies dans le fichier *variables.tfvars* :

```
# Region
region = "GRA7"

# Network - Private Network
network = {
 name = "myNetwork"
}

# Network - Subnet
subnet = {
 name       = "mySubnet"
 cidr       = "192.168.12.0/24"
 dhcp_start = "192.168.12.100"
 dhcp_end   = "192.168.12.254"
}

# Network - Router
router = {
 name = "myRouter"
}
```

C'est le sous-réseau qui porte les informations telles que l'adresse de la gateway, les serveurs DNS et les informations DHCP.

La création d'un routeur se fait en deux étapes. La première est la création du routeur elle-même, qui s'interface automatiquement sur le réseau public **Ext-Net** et obtient ainsi son adresse IP publique, et la deuxième étape est la création de la ressource *interface*, dont le rôle est de connecter le routeur au réseau privé. C'est tout ce qu'il y a à faire pour la partie réseau de cet exemple.

Passons maintenant à la suite, c'est-à-dire l'instance virtuelle que nous allons déployer dans ce réseau. Bien que connectée exclusivement au réseau privé, cette instance devra être accessible. Joignable d'une part par le réseau public, d'où l'emploi d'un Floating IP, et accessible d'une autre part via le protocole SSH.

Avant de pouvoir déployer notre instance, nous devons au préalable créer un jeu de clé SSH.

Jeu de clé SSH **Figure 3**

La seconde étape sera donc dédiée à la création d'un jeu de clé SSH. Un jeu de clé SSH se compose de deux clés, une dite privée (private key) qui se situe du côté client, ou utilisateur, et une autre dite publique (public key), qui sera diffusée sur les machines distantes.

Le jeu de clé (keypair) est directement créé dans l'environnement OpenStack, toujours via le provider de ce dernier.

Le module proposé *ssh_keypair* permet toutefois la création de jeux de clés dans plusieurs régions. Dans ce cas, le couple de clés privée et publique est créé dans une région choisie, et la clé publique est ajoutée dans les autres régions listées dans la variable *to_reproduce_regions*.

On appelle le module dans le fichier main.tf tel que :

```
module "keypair" {
 source = "../../modules/ssh_keypair"
```

```
 keypair = var.keypair
}
```

La variable keypair est définie dans le fichier variables.tf tel que :

```
variable "keypair" {
 description = "Keypair"
 type = object({
  name          = string
  main_region       = string
  to_reproduce_regions = list(string)
  keys_path       = string
 })
}
```

Dans notre exemple, nous ne travaillons que sur une seule région, seuls les attributs *name* et *main_region* sont à renseigner.

```
keypair = {
 name          = "myMainKeypair"
 main_region       = "GRA7"
 to_reproduce_regions = []
 keys_path       = "."
}
```

Notre jeu de clé est prêt à être généré, nous avons toutes les informations nécessaires pour déployer l'instance.

Instance bastion **Figure 4**

La troisième étape est la création d'une instance virtuelle, ici nommée *bastion*, car son rôle est simplement d'offrir un moyen de connexion avec l'environnement privé.
Cette machine est de type Linux (ici une distribution Ubuntu), on lui indique de se connecter au réseau privé, et on y déploie la clé SSH nouvellement créée.
L'appel à ce module se fait toujours via le fichier main.tf :

```
module "bastion" {
 source   = "../../modules/instance_simple"
 depends_on = [module.keypair, module.network]
 instance  = var.bastion
}
```

et sa variable principale est la variable bastion, définie dans le fichier variables.tf tel que :

```
variable "bastion" {
 description = "Bastion Parameters"
 type = object({
  region     = string
  network_name = string
  keypair_name = string
  name       = string
  flavor     = string
  image      = string
  user       = string
 })
}
```

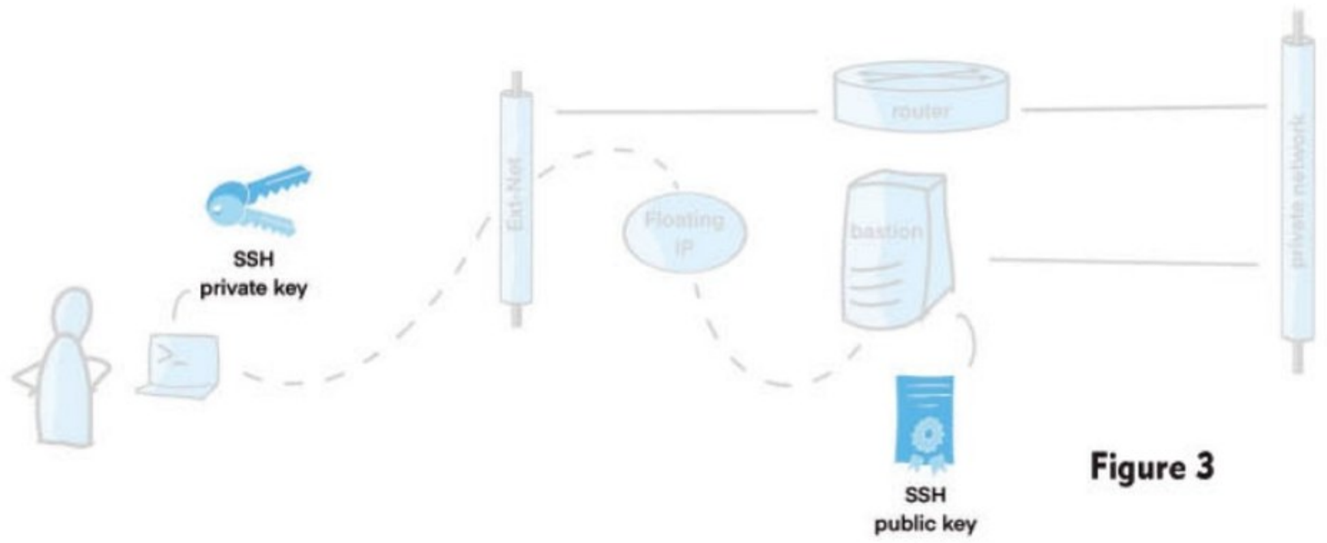


Figure 3

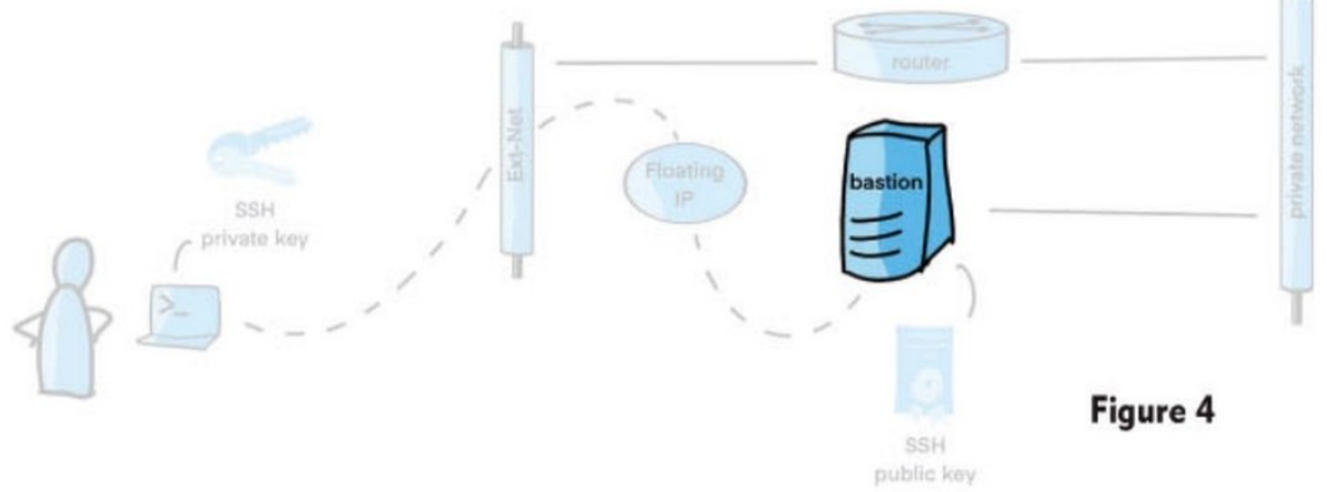


Figure 4

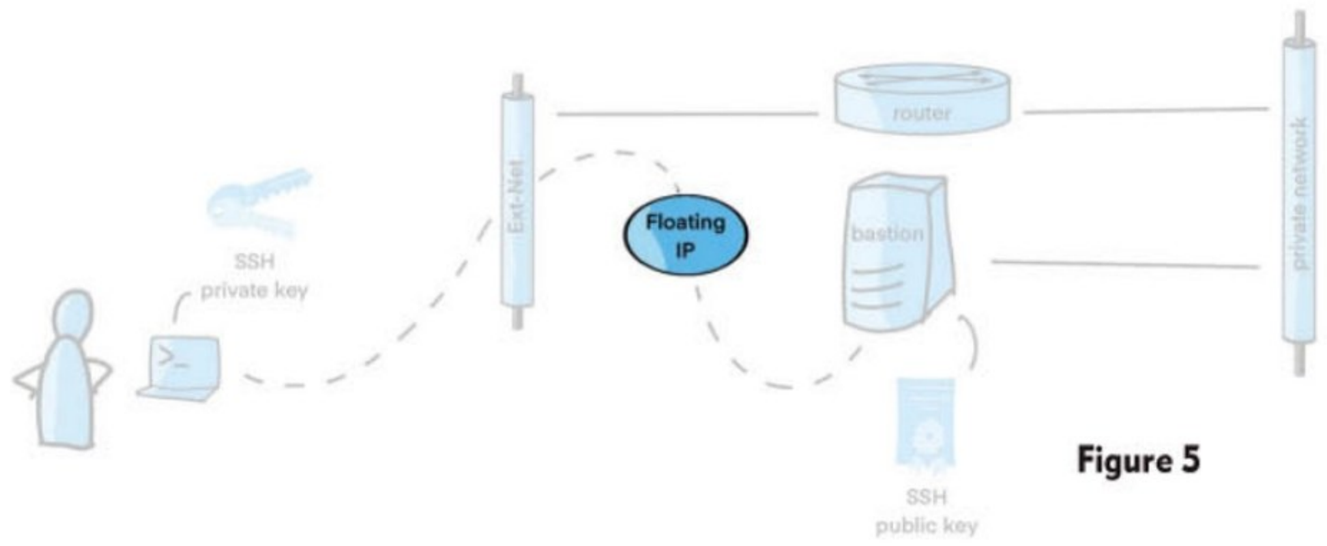


Figure 5

et valorisée via le fichier variables.tfvars tel que :

```
bastion = {
 region     = "GRA7"
 network_name = "myNetwork"
 keypair_name = "myMainKeypair"
 name       = "bastion"
 flavor     = "b2-7"
 image      = "Ubuntu 20.04"
 user       = "ubuntu"
}
```

Ici nous ne faisons que reporter les valeurs des ressources créées précédemment, comme le nom du réseau, le nom de la clé SSH, de la région. Nous y ajoutons les paramètres spécifiques à l'instance, comme le type de machine (*flavor*), le système d'exploitation (*image*), le nom de l'utilisateur principal (*user*) et bien sûr le nom de l'instance(*name*).
À ce moment, notre instance n'a qu'une interface réseau, qu'on a spécifiquement demandé de se connecter au réseau privé. Il faut mettre en place la mécanique pour pouvoir y accéder.

Attention

Suite du dossier dans PROGRAMMEZ! 257

Heroku payant : quelles alternatives pour déployer ses POC ?



Antoine Garnier
Consultant R&D DevOps



Paul Barrié
Consultant Devops

Les Filles & Les Garçons DE LA TECH

En novembre dernier, le tiers gratuit d'Heroku, permettant de déployer des services (dynos) légers, est devenu payant, passant à 7 € par mois. Une mauvaise nouvelle pour tous les développeurs qui utilisaient cette solution PaaS de Salesforce pour déployer leurs petits services Webs et autres sites statiques.

Bien qu'Heroku dise au revoir à la gratuité, tout n'est pas perdu puisque des alternatives existent pour déployer de petites applications gratuitement avec une facilité équivalente à la solution proposée par Heroku.

Pour les lecteurs de Programmez, nous avons testé plusieurs solutions PaaS afin d'y voir un peu plus clair dans l'écosystème existant et de permettre aux développeurs de déployer leurs petits projets. Pour notre démonstration, nous avons déployé deux applications simples :

- Une API en Go contenant une seule route simple en GET qui renvoie juste une chaîne de caractères.
- Une app React qui consomme cette API Go.

Si vous voulez faire des tests par vous-mêmes, vous pourrez retrouver ces applications dans le repository `dummy-app` des Filles et des Garçons de la Tech.

Render.com : un concurrent sérieux à Heroku

La première PaaS qui a attiré notre attention est la jeune entreprise californienne Render.com. Créée en 2018, cette plateforme propose un service assez proche d'Heroku, avec un plan gratuit qui permet de remplir facilement notre cas d'usage.

En effet une large palette de services peut y être déployée : des sites statiques ou des services type API bien sûr, mais aussi du cache Redis, des Cron Jobs ou encore des consumers pour des queues de messages.

L'offre freemium permet d'utiliser les services de Render.com avec les limites suivantes :

- Le type de machine est limité à 500Mo de RAM et 1 CPU partagé.
- La bande passante est limitée à 100 Go.
- Le temps de build est limité à 500 minutes par mois.
- L'impossibilité de travailler en équipe sur un même projet.
- Un support limité à la communauté et au mail avec les équipes de Render.com.

Si vous souhaitez opter pour cette solution, il vous faudra néanmoins faire attention à ne pas dépasser les quelques limites imposées par le plan gratuit au risque de voir la facture augmenter : 30 $ par tranche de 100 Go au-delà des 100 Go gratuits et 5 $ pour 1000 minutes de build au-delà des 500 minutes offertes.

Le déploiement de vos applications est assez intuitif et se fait assez directement via l'interface graphique proposée par la plateforme.

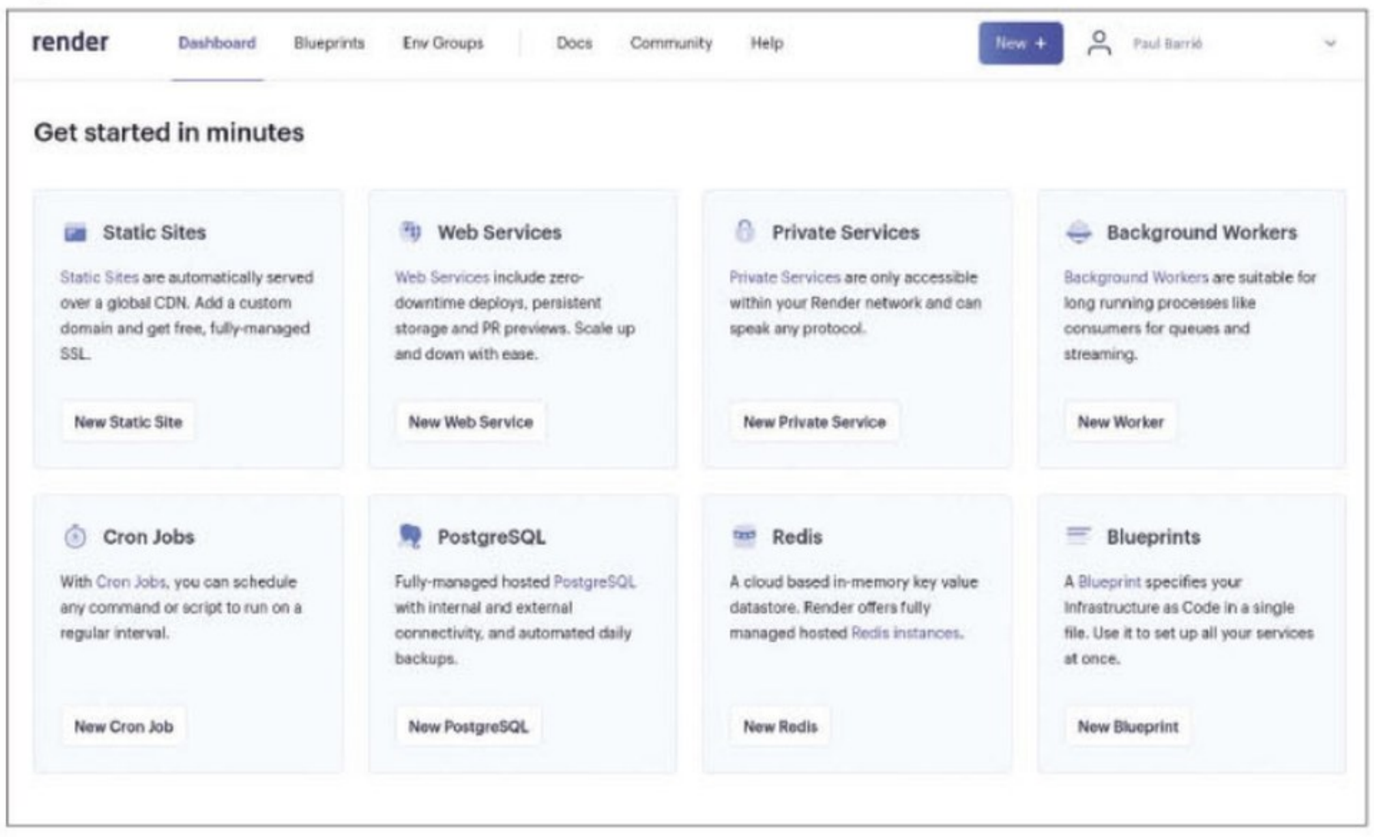
Une fois inscrits et connectés sur la plateforme, vous arrivez directement sur un dashboard vous proposant de déployer votre première application : **Figure 1**

Nous allons créer un premier service pour notre API. Pour cela, nous allons dans l'onglet "Web Services". Il nous est par la suite proposé de choisir le repository dans lequel est située votre application. S'il est privé, vous devrez alors vous connecter soit à Gitlab, soit à GitHub. S'il est public, vous aurez juste à rentrer l'URL du repository. On peut néanmoins regretter l'impossibilité de se connecter à d'autres VCS (comme Bitbucket) en fournissant par exemple une clé SSH pour permettre l'accès au repository.

On vous demandera ensuite :

- Le nom que vous souhaitez donner à votre service (dans notre cas `dummy-api` puis `dummy-app`).
- La région dans laquelle vous souhaitez déployer votre service. Quatre régions sont disponibles : US West, US East, Europe Centrale et Southeast Asia. On choisit naturellement la région européenne.
- La branche sur laquelle Render.com doit aller chercher l'application.
- Un éventuel sous-dossier qui constitue la racine de votre application (dans notre cas `dummy-api` puis `dummy-app`).
- Le runtime utilisé : dans notre cas ce sera Node pour l'app et Go pour l'API.
- La commande de build qui, hors cas particulier, devrait être convenablement générée par les informations préalablement fournies.
- La commande de run qui elle aussi devrait être générée en fonction des informations renseignées.

Figure 1



- Et le type d'instance que l'on souhaite déployer. On choisit le premier type qui est gratuit (parce qu'on aime quand c'est gratuit).
- Un onglet en bas de la page, "Advanced", permet de fournir des informations supplémentaires telles que :
  - Des variables d'environnement, qui peuvent être fournies directement ou via un fichier .env.
  - Un chemin permettant de vérifier la bonne santé de votre application.
  - Une option d'"auto-deploy" qui va permettre de recharger le service à chaque push sur la branche et le repository fourni.

Vous pouvez ensuite confirmer les informations rentrées. Vous arrivez alors sur une nouvelle page et une fois le build terminé et l'application déployée, vous pouvez accéder à votre service via l'URL située en haut à gauche. **Figure 2**
Pour le déploiement de l'application, il vous suffit de choisir l'option "Static Site" dans l'onglet "New", et de fournir les informations demandées comme pour l'API en précisant la commande de build et la localisation des fichiers statiques devant être servis. **Figure 3**

En revenant sur la page de dashboard vous verrez l'ensemble des services que vous avez déployés. **Figure 4**

Si vous avez fourni de mauvaises informations lors de la création du service, vous pourrez toujours les modifier via l'onglet "Settings" en cliquant sur le service en question dans le dashboard.
Si malgré toutes vos modifications, vous n'arrivez toujours pas à démarrer votre service, vous pouvez vous connecter en SSH au service Web en haut de la page dédiée à ce dernier. **Figure 5**
Vous pouvez également accéder à l'ensemble des logs (et effectuer une recherche sur ces derniers) dans l'onglet dédié du service.
Si vous souhaitez par ailleurs automatiser le déploiement et la mise à jour de vos services, vous pouvez également utiliser le nouveau CLI développé par Render.com et disponible sur https://github.com/render-oss/render-cli. Attention néanmoins à ne pas dépasser les 500 minutes de build mensuel, sous peine d'être facturé. Néanmoins, n'ayant pas besoin de fournir une carte bancaire valide, vous ne serez pas facturé en cas de dépassement de l'offre gratuite.

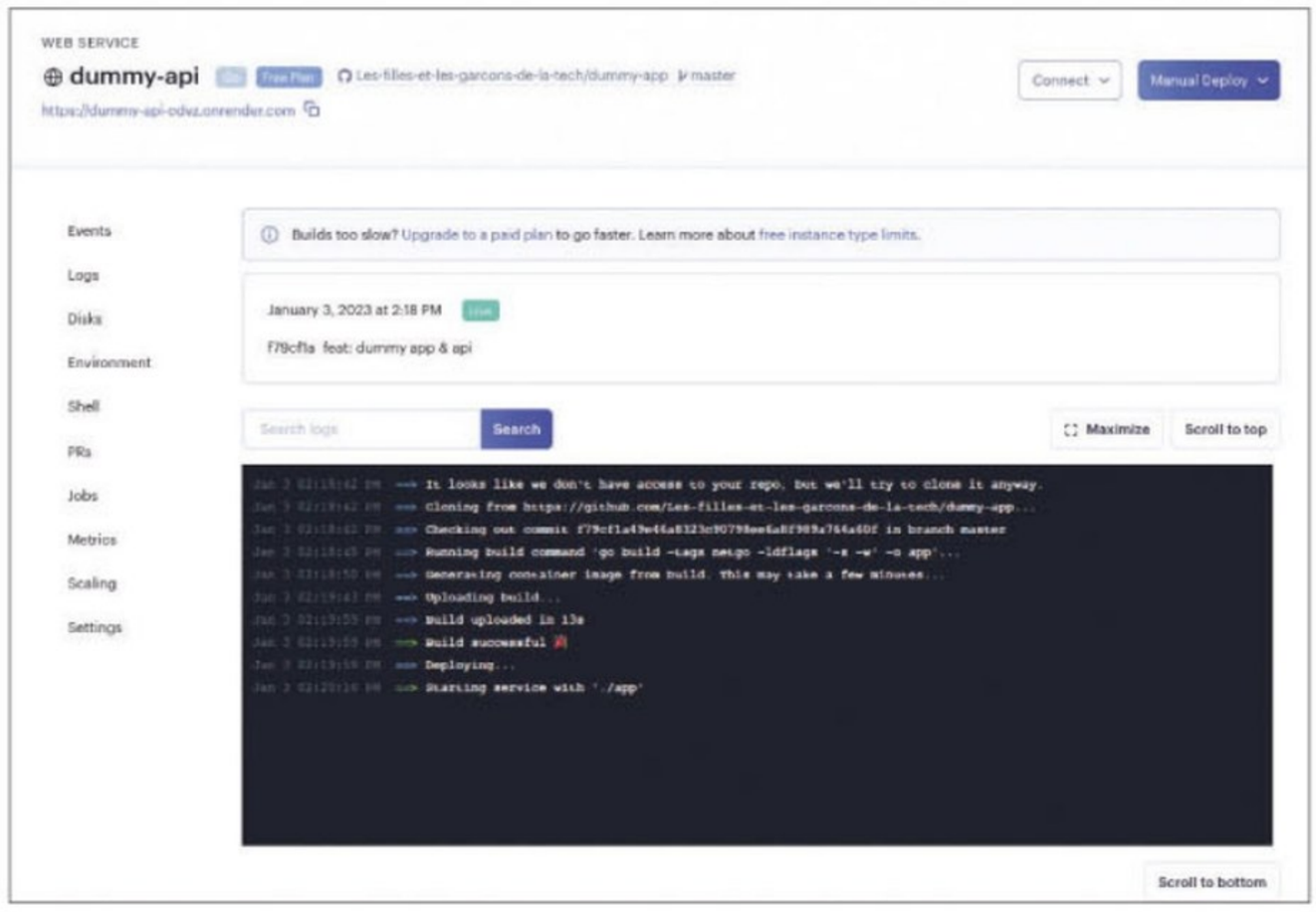
Render.com	■ Une large gamme de service proposée ■ Un bon feed-back utilisateur	■ Un service qui devient rapidement cher dès que l'offre gratuite est dépassée. ■ L'absence de connecteur à un VCS autre que GitHub ou GitLab

Fly.io : un PaaS en ligne de commande

Fly.io est une solution PaaS du même type que Render.com qui propose un plan freemium ("Hobby plan") qui inclut :

- 3 instances avec 1 CPU partagé

Figure 2



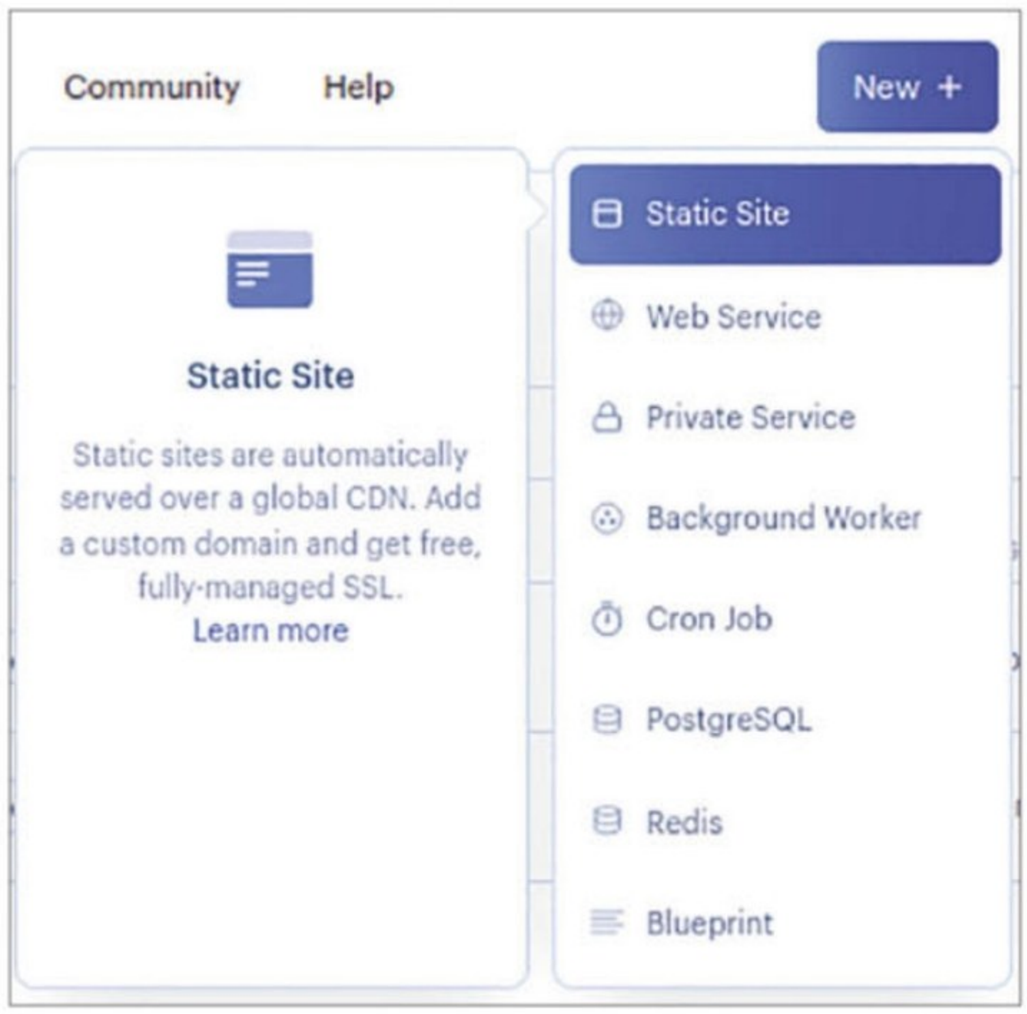


Figure 3



Figure 4

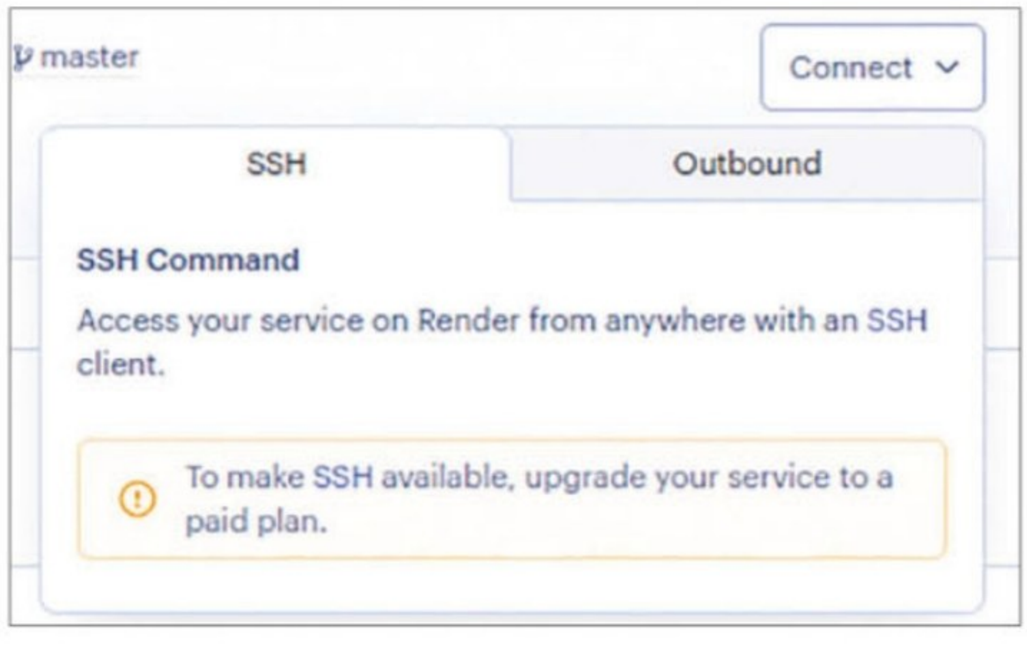


Figure 5

```
└ [2023-01-03 15:57:19] $ >>> flyctl launch
Creating app in /home/paulb/Documents/fgtech/articles/programez/dummy-api
Scanning source code
Detected a Go app
Using the following build configuration:
        Builder: paketobuildpacks/builder:base
        Buildpacks: gcr.io/paketo-buildpacks/go
? Choose an app name (leave blank to generate one):
automatically selected personal organization: Paul Barrié
? Choose a region for deployment: Paris, France (cdg)
Created app muddy-mountain-907 in organization personal
Admin URL: https://fly.io/apps/muddy-mountain-907
Hostname: muddy-mountain-907.fly.dev
Wrote config file fly.toml
? Would you like to set up a Postgresql database now? No
? Would you like to set up an Upstash Redis database now? No
? Would you like to deploy now? Yes
==> Building image
```

Figure 6

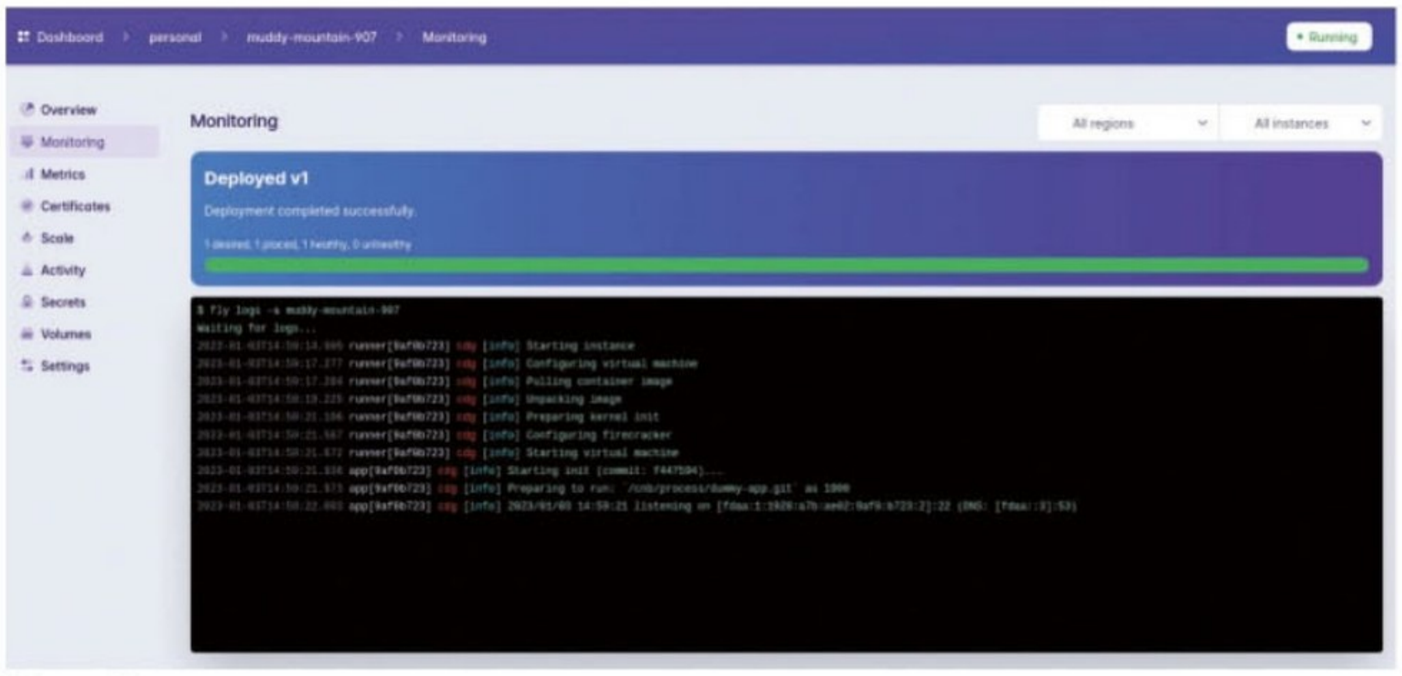


Figure 7

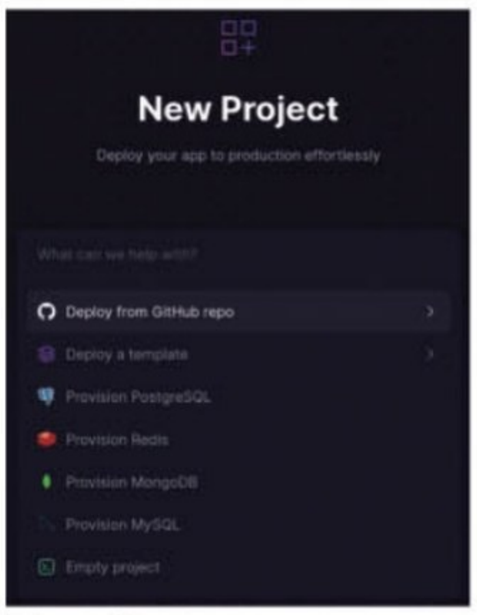


Figure 8

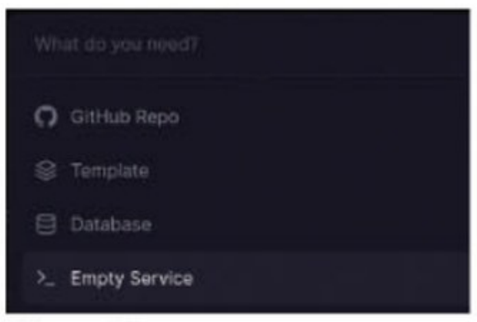


Figure 9

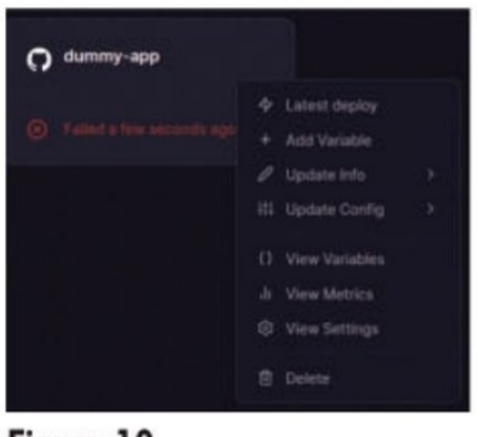


Figure 10

- 3 Go de stockage
- 160 Go de transfert de données

Contrairement à Render.com, Fly.io a fait le choix de faire passer les commandes de déploiement directement via un CLI, `flyctl`. Leur site internet sert uniquement à l'inscription et à la documentation.

Installez le CLI en suivant les instructions selon votre machine dans https://fly.io/docs/hands-on/install-flyctl/.

Vous pouvez alors créer un compte si vous n'en disposez pas d'un :

```
flyctl auth signup
```

Vous serez alors redirigé vers une page Web et il vous suffira de suivre les instructions. Si vous avez déjà un compte, il vous suffit de vous connecter :

```
flyctl auth login
```

Pour déployer votre application, il vous faudra ensuite vous rendre à la racine de l'application que vous souhaitez déployer et lancer la commande :

```
flyctl auth login
```

Une série d'informations vous sera alors demandée : **Figure 6**

Une fois toutes les informations fournies, vous devrez voir apparaître un fichier fly.toml qui servira pour Fly.com à déployer votre application.

Vous pouvez le modifier pour par exemple ajouter des variables d'environnement.

En entrant la commande `flyctl apps list` vous pourrez voir l'ensemble des applications et leur statut.

Si les informations contenues dans le fichier toml vous conviennent, vous pouvez alors déployer votre application en entrant la commande `flyctl deploy` à la racine de votre application.

Flyctl va alors créer une image docker, la pousser dans un registry et la déployer. À la fin, une URL redirigeant vers le dashboard du service déployé est affichée. **Figure 7**

Vous y verrez les logs du service déployé et pourrez configurer plus finement le déploiement de ce-dernier.

Render.com	■ Un CLI plutôt complet	■ Une offre freemium plus limitée que chez Render.com. ■ Une configuration moins fine avec une typologie de services limitée. ■ Un feed-back utilisateur parfois absent dans l'utilisation du CLI.

Railway

Railway fournit un plan gratuit mensuel. Il n'est cependant pas à la hauteur de ce que d'autres plateformes proposent. Une fois vérifié, votre compte vous donne accès à $5 de crédits ou 500h d'exécution, ce qui ne sera pas suffisant pour laisser tourner une instance en permanence.

Railway vous propose plusieurs types de services. Des bases de données SQL et orientées document avec MongoDB, du cache Redis ou des applications que vous avez vous-même développées, ce qui nous intéresse particulièrement dans notre cas. **Figure 8**

Trois solutions s'offrent à vous pour déployer votre application :

- À partir d'un template, où vous sera proposée une liste pléthorique de possibilités qui recouvre du React/Vite, du FastAPI, du Django ou même un blog géré par Railway. Néanmoins, un certain nombre de types d'applications peuvent manquer à l'appel : pas de Spring et du Go déployable qu'en utilisant le framework GoFiber.
- En créant un projet vide dans lequel vous allez pouvoir déployer un ensemble d'applications.
- À partir d'un repository GitHub.

Nous créons alors un nouveau projet. Dans celui-ci, on nous propose d'ajouter un service soit à partir d'un repository GitHub, d'un template, d'une base de données ou de créer un service vide.

Créer un service vide n'est pas vraiment utile puisqu'il vous faudra dans tous les cas connecter un repository GitHub contenant votre application. **Figure 9**

On choisit donc de créer un nouveau service à partir d'un repository GitHub, on nous propose de choisir le repository où est située l'application que nous souhaitons déployer. Il va alors essayer immédiatement de déployer l'application, ce qui va échouer.

Vous pouvez alors soit cliquer sur la carte et vous rendre dans les paramètres soit vous y rendre en utilisant le clic droit.

Figure 10

Vous pouvez alors préciser l'ensemble des informations relatives à votre déploiement telles que le répertoire racine de votre application, les commandes de build et de run, les healtchchecks, etc. Il y a également un champ "Watch Paths" dans lequel on peut préciser les dossiers qui vont

entraîner un redéploiement lors de leur modification.
À chaque validation de la modification d'un champ, un nouveau déploiement est enclenché. Un domaine est par ailleurs attribué à votre service. Il vous faudra néanmoins l'activer en cliquant sur le champ "Enable". Vous pourrez également ajouter des variables d'environnement dans un onglet prévu à cet effet.
En cliquant sur le champ *Deployments*, vous pourrez voir l'ensemble des builds réalisés ainsi que les logs des services.

Railway	■ Une interface et un parcours utilisateur plutôt agréables même si quelques fonctionnalités sont assez peu ergonomiques.	■ Une offre freemium insuffisante pour déployer au moins une instance en continu.

Adaptable.io : jeune solution émergente.

Moins avancée que les outils précédents, Adaptable.io est une jeune plateforme de déploiement de projets dont le lancement de la bêta a eu lieu le 24 juin 2022. Sur cette plateforme, les utilisateurs pourront déployer une assez large palette de services allant de l'API au frontend en passant par des services socket.
L'outil propose un tiers gratuit incluant le déploiement et la maintenance d'un conteneur par application et des bases de données ainsi que la CI depuis GitHub.
Le tier gratuit est cependant limité par un conteneur par application ainsi qu'1 CPU, 256 Mo de RAM et 1 Go de stockage. L'accès reste cependant contraignant à ce jour, car il vous faudra demander un token d'accès à la bêta pour pouvoir accéder au service (obtenable par demande via chat ou mail avec néanmoins une réponse relativement rapide).
Dans le cas où l'utilisateur souhaite aller plus loin avec cet outil, il est également proposé des tiers payants individuels ou pour des équipes, dont les prix varient en fonction du bouquet choisi et facturant chaque application par mois (7 à 20 $/app/mois pour les plans individuels et 50 $/app/mois ou prix customisé pour les équipes et entreprises avec plus de soutien technique et de suivi des projets par l'équipe d'Adaptable).

La plupart des langages et frameworks utilisés pour les applications Web sont supportés par l'outil (Go, Python, JavaScript, Ruby, Feathers, Prisma, Nestjs, React, Ruby on Rails, etc.) ainsi que plusieurs types de bases de données (SQL ou NoSQL).
Dans le cas où les frameworks et langages utilisés sont supportés par l'outil, les utilisateurs pourront profiter de la facilité et de l'ergonomie de déploiement que propose l'outil.
Le déploiement de l'application se fait à partir d'un répertoire GitHub cible, lors du déploiement de l'application par Adaptable, des pipelines de CI sont mis en place afin d'assurer une mise à jour de l'application dès que du code est poussé sur la branche main/master du répertoire. Ce fonctionnement simplifiera la vie des développeurs qui ne souhaitent pas s'occuper de cette partie intégration continue.
D'autre part, pour les développeurs qui souhaiteraient débuter rapidement un essai sur la plateforme, des exemples d'applications prêtes à déployer sont disponibles au nombre de 4 pour le moment.
Parmi ces templates, on retrouve :

- Deux applications de chat : l'une développée en Feather.js et fonctionnant avec du MongoDB en backend, l'autre avec du PostgreSQL en backend.
- Deux applications de ToDo en ligne, l'une développée en NestJS avec Mangoose, l'autre en Express.js avec Prisma ORM. Les données sont persistées dans une base de données MongoDB.

La première étape consiste à renseigner le répertoire GitHub cible à partir duquel l'application sera déployée. **Figure 11**
Il faut ensuite choisir un template parmi Feather, NestJS, NodeJS, Flask, Python et Express en fonction du service à déployer. Le type de technologies est donc assez restreint à un nombre limité de frameworks de l'écosystème Node et Python. **Figure 12**
L'outil propose également de déployer une base de données si l'application le nécessite. Il est ensuite possible de spécifier à l'aide d'un script la manière dont votre application sera buildée et lancée. **Figure 13**
Après une dernière vérification du contenu du répertoire, le déploiement de l'application peut être lancé. Une fois l'application déployée, il est possible d'accéder aux logs de build et de run de l'application via les différents onglets proposés.

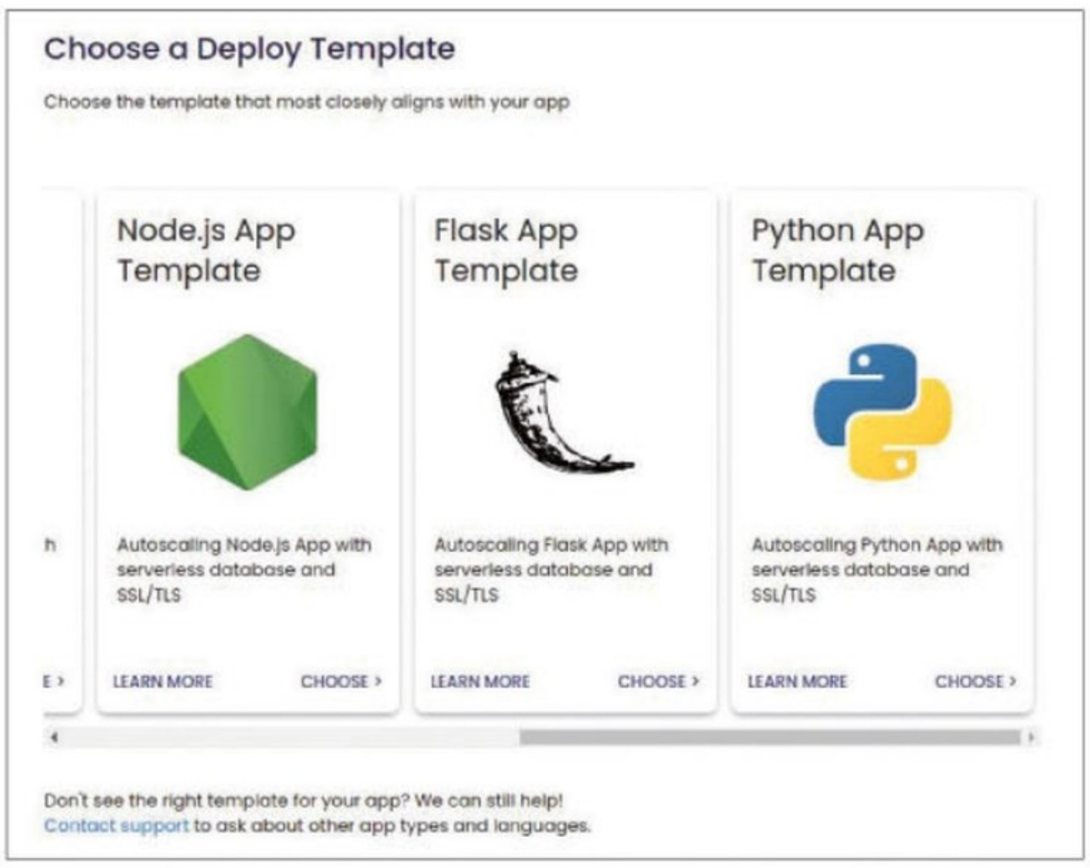


Figure 12

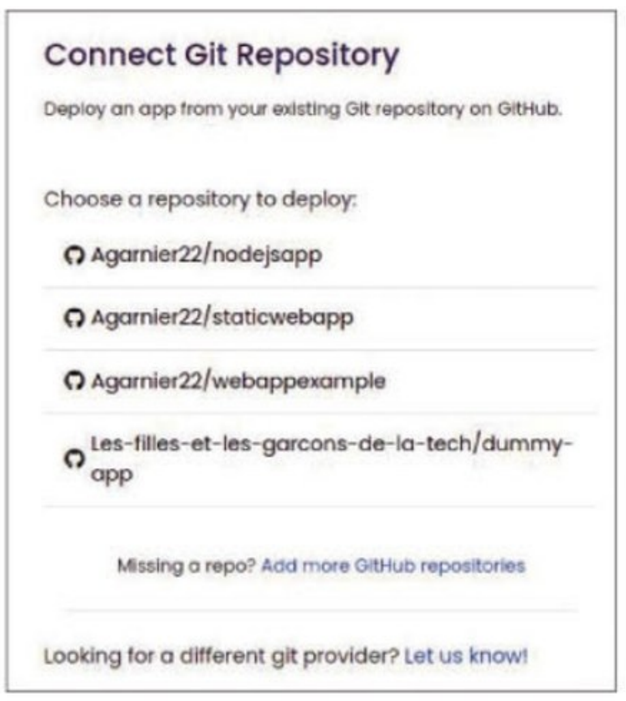


Figure 11

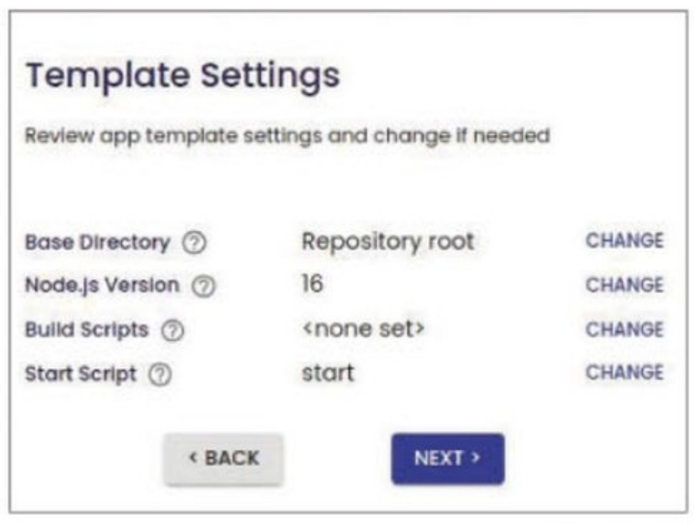


Figure 13

Adaptable.io	■ Déploiement simple et intuitif (si technologie supportée)	■ Technologies et formats d'applications limitées.

Les services PaaS de DigitalOcean

Avec plus de 10 ans d'existence, le fournisseur de Cloud new-yorkais propose désormais des solutions PaaS. Elles permettent en outre de déployer et de gérer ses applications de façon simple et intuitive à travers un *Serveur Privé Virtuel* (VPS) aussi nommé *Droplets*, et accessible via une interface Web ou un CLI. A l'instar des fournisseurs cloud tels qu'AWS, Azure ou GCP, la plateforme IaaS proposée par DigitalOcean permet de paramétrer ses infrastructures à sa convenance en proposant différentes régions d'hébergement, différentes tailles de machines et des services additionnels permettant de suivre et de gérer au mieux ses applications.

Concernant la tarification, il existe un **tiers gratuit, mais qui ne concerne que les applications statiques et limite le nombre d'applications à trois**, seuil au-delà duquel chaque application supplémentaire sera facturée 3 $ par mois.

La solution proposée par DigitalOcean est donc intéressante d'un point de vue performances, mais reste limitée dans le cadre du tiers gratuit.

La plateforme propose un service "applications" capable de détecter les configurations requises et de construire et déployer toute l'infrastructure d'une application. Pour les applications simples comme de petites preuves de concept, ce service peut être d'une grande efficacité. Par ailleurs, son utilisation est relativement simple.

Dans un premier temps, il faut créer un projet et une nouvelle application (ou tout autre service) dans ce projet. Dans notre cas, on choisira le service Apps.

Dans le cas du déploiement d'une application, il faudra ensuite renseigner le répertoire dans lequel se trouve l'application à déployer ou l'image de l'application en question.

Il est également possible de choisir parmi de nombreuses applications prêtes à l'emploi proposées par la plateforme (allant du frontend avec du React ou Vue.js au backend en Golang, Python ou Ruby par exemple)

Une fois le répertoire choisi ainsi que le dossier dans lequel se trouve l'application, le service apparaît dans la liste des services à déployer. Il est alors possible de rajouter des services, par exemple si un site Web possède un frontend et un backend développés sur le même répertoire, il suffit de pointer chaque service sur la bonne partie du répertoire.

On choisit ensuite le mode de facturation qui sera associé à chaque service ajouté dans l'onglet "Edit Plan". L'étape suivante consiste à ajouter les éventuelles variables d'environnement nécessaires au bon fonctionnement des services et de l'application dans sa globalité.

Dans la partie "Info", il est possible de modifier les informations liées à l'application telles que le nom, le projet associé ainsi que la région de déploiement ciblée.

Enfin une partie review permet de récapituler les informations entrées pour le déploiement de l'application et de ses services, ainsi qu'une évaluation du coût mensuel des ressources déployées.

Il est possible de voir les logs de construction et de déploiement de l'application pendant son lancement ainsi que pendant son fonctionnement. Ainsi il est possible d'étudier dans le détail les éventuelles erreurs rencontrées et de les résoudre. Il est également possible de suivre sur la page de l'application certaines données d'utilisation telles que le CPU et la RAM en plus des informations de statut.

DigitalOcean	■ Une interface et un parcours utilisateur intuitifs et ergonomiques	■ Free tiers limité à 3 applications statiques (3 $/mois/application supplémentaires)
	■ Informations de déploiement claires et accès simple aux conteneurs déployés (terminal d'accès intégré)	■ Pas de free tiers pour les autres types d'applications (serveurs, api, etc.)

Conclusion

On espère avoir pu rassurer les développeurs de Programmez : les alternatives à Heroku sont assez nombreuses ! Pour les plus téméraires d'entre vous qui souhaitent gérer leurs déploiements avec un niveau plus bas, il vous est bien sûr possible d'utiliser les IaaS des principaux cloud providers du marché (AWS, Azure, GCP) qui proposent également un accès en tiers gratuit à leurs services.

Pour ceux voulant rester sur du PaaS, il convient de choisir les outils adaptés à vos besoins. Si vous recherchez un outil capable de s'intégrer dans une CI, il serait intéressant d'utiliser les outils possédant une interface en ligne de commande comme Fly.io ou des outils intégrant un système de mise à jour automatique en fonction des changements sur le repository comme Railway.

Si vous recherchez une expérience utilisateur simple, il semble plus pertinent de choisir des outils tels que Render.com, DigitalOcean voire Adaptable. Nous avons en effet particulièrement apprécié l'utilisation de Render ou Digitalocean. Enfin, dans le cadre d'une utilisation pour de simples preuves de concept ou d'essai de déploiement, il convient de choisir un outil proposant une tarification intéressante. Pour ce dernier point, chaque outil a sa propre tarification et son offre gratuite plus ou moins avantageuse selon les types d'applications déployées. Parmi les solutions testées, Render.com est à la fois celle qui permet de faire face à la plus large palette de cas d'usages tout en offrant un tiers gratuit vraiment intéressant.

L'avenir de l'IA est-il dans le cloud ?

Kevin Ansard
Ingénieur d'Études et Développement
Neos-SDI

Depuis plusieurs années, l'intelligence artificielle devient un facteur important au développement des secteurs déjà implantés et en cours de développement pour qu'ils deviennent durables dans le temps. Si nous prenons un exemple d'actualité comme ChatGPT, il est facile de dire que son utilité n'est plus à prouver et que l'usage ne se limite pas à l'IT. Cette démocratisation apporte son lot de nouvelles problématiques. En tant que développeurs, nous sommes de plus en plus invités à comprendre et à mettre en place des intelligences artificielles. Cependant, la mise en place d'IA n'est pas accessible à tous les dévs. Cela implique de prendre le temps de se former et bien comprendre son fonctionnement.

L'étendue des possibilités de l'IA est pratiquement infinie si l'on rapproche cela à des logiques métiers. Les notions fondamentales à comprendre sont nombreuses :

- Les réseaux de neurones
- Le DeepLearning
- Le Maching Learning
- Les modèles
- Les matrices
- Les dérivés
- Etc.

Tous ces points ne sont pas simples à appréhender. De plus, le coût de formation en temps et en budget sont intimement liés à la complexité de l'IA. De nombreux fournisseurs de cloud public proposent des services IA, par exemple, Microsoft avec les Azure Cognitive Services. Est-ce que l'avenir de l'IA se trouve dans le cloud (public) ?

Problématique : le quoi et le comment

Pour illustrer mon propos, nous allons prendre un projet concret que j'ai réalisé sur Azure : un client avait une problématique claire : comment automatiser le tri de ces pièces jointes par secteur d'activité (RH, IT, COM, etc.) ?

La réponse qui paraissait la plus évidente était de déployer une IA et cela a d'ailleurs été ma première idée, mais comment ? Comment mettre en place une IA performante pour trier plusieurs centaines de pièces jointes par jour, sachant que je n'ai jamais fait d'IA ?

Ce POC était censé prendre un mois. Je n'avais donc pas le temps de réaliser un projet abouti ni le courage de le développer. Essayons de pousser plus loin la réflexion. Si je devais m'atteler à apprivoiser les notions mathématiques derrière une IA, cela me prendrait au minimum un mois et ce n'est que la première étape vers la création d'une IA opérationnelle. Même si ces notions étaient assimilées, il m'aurait fallu plusieurs mois pour la mettre en place dans mon code.

La réponse s'est forgée lors d'une discussion avec l'un de mes mentors. Pourquoi ne pas utiliser Azure et plus précisément Azure Cognitive Services ?

Azure Cognitive Services

Azure Cognitive Services est un ensemble de services sur Azure. Son but est de fournir clé en main des modèles, et services, d'IA aux développeurs. Azure Cognitive Services se découpe en 5 grandes catégories, chacune de ces catégories correspond à un type d'usage :

- Langage pour le texte
- Vision pour l'analyse d'image et de vidéos (compute vision)
- Décision pour des décisions arbitraire
- Speech pour la reconnaissance vocale
- OpenAI **Figure 1**

Ces catégories regroupent plusieurs services. Celui qui nous intéresse aujourd'hui est le Text Classification qui fait partie de la catégorie Langage. Text Classification permet de déduire des labels prédéfinis en fonction du texte analysé. **Figure 2**

Si nous revenons à notre contexte initiale, classé des pièces jointes de mail en fonction de leur appartenance à un service (RH, IT, COM, etc.) cela semble parfait. En effet en récupérant le texte contenu dans le mail ainsi que dans la pièce jointe nous pourrons en déduire son label (sa catégorie). Je prends donc ce chemin, même si je l'admets, j'étais dubitatif quant à l'efficacité de ce service. Pour faciliter la mise en place de notre PoC, nous allons nous baser sur le sample qu'Azure (NDLR : toujours appliquer le principe du RTFM).

Mise en place du projet

La mise en place d'un service cognitif Azure, deux solutions sont possibles :

- La première est d'utiliser le portail d'Azure ainsi qu'un outil appelé Language Studio.
- La seconde est d'utiliser l'api d'Azure. Pour des raisons de simplicité, nous verrons aujourd'hui la première implémentation.



Figure 1

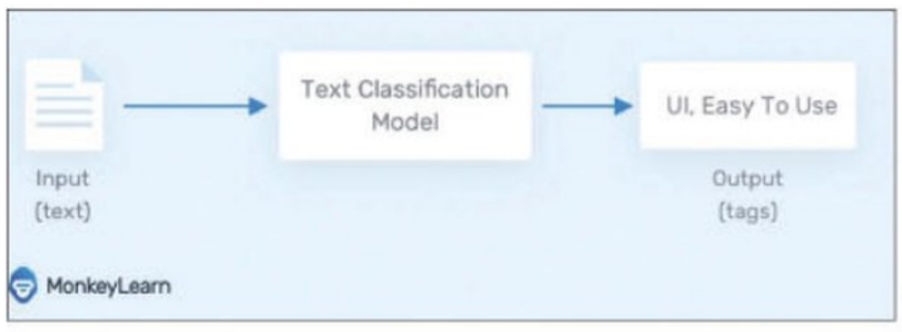


Figure 2

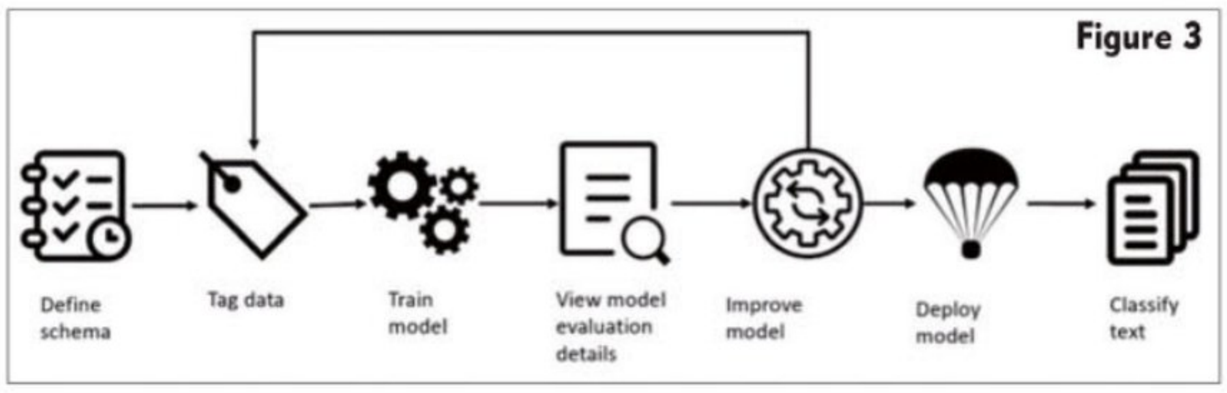


Figure 3

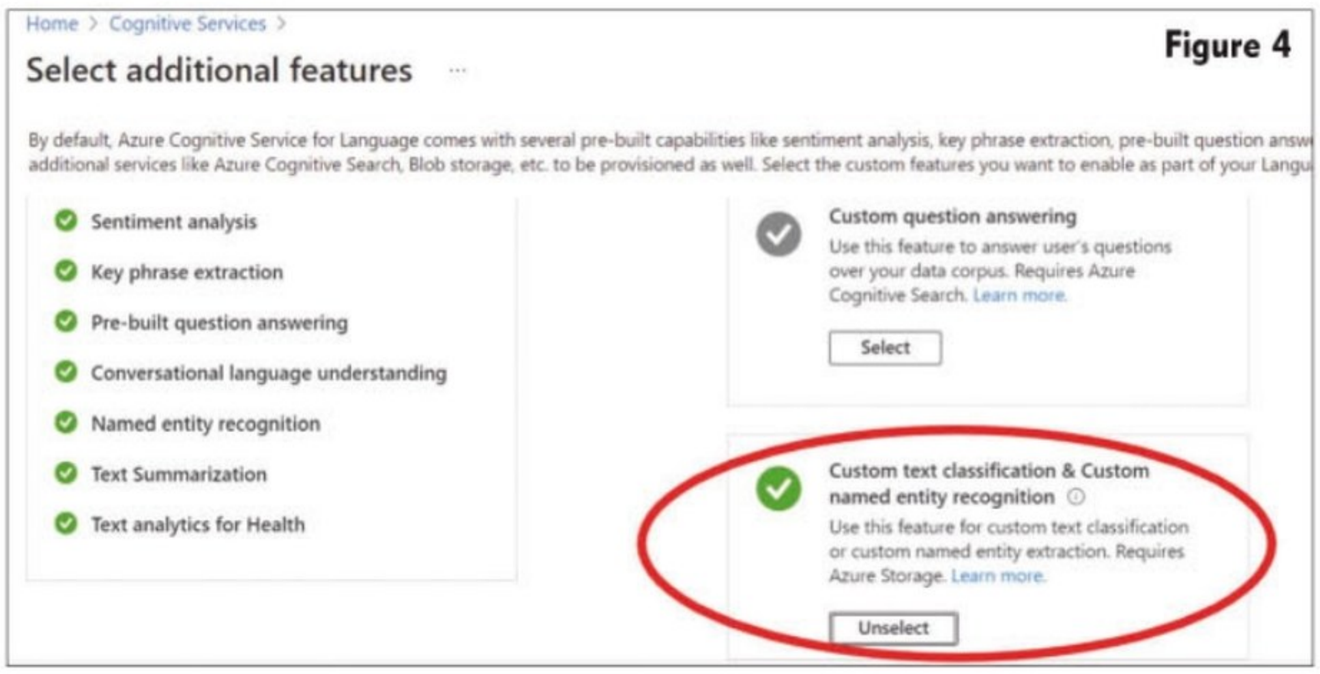


Figure 4

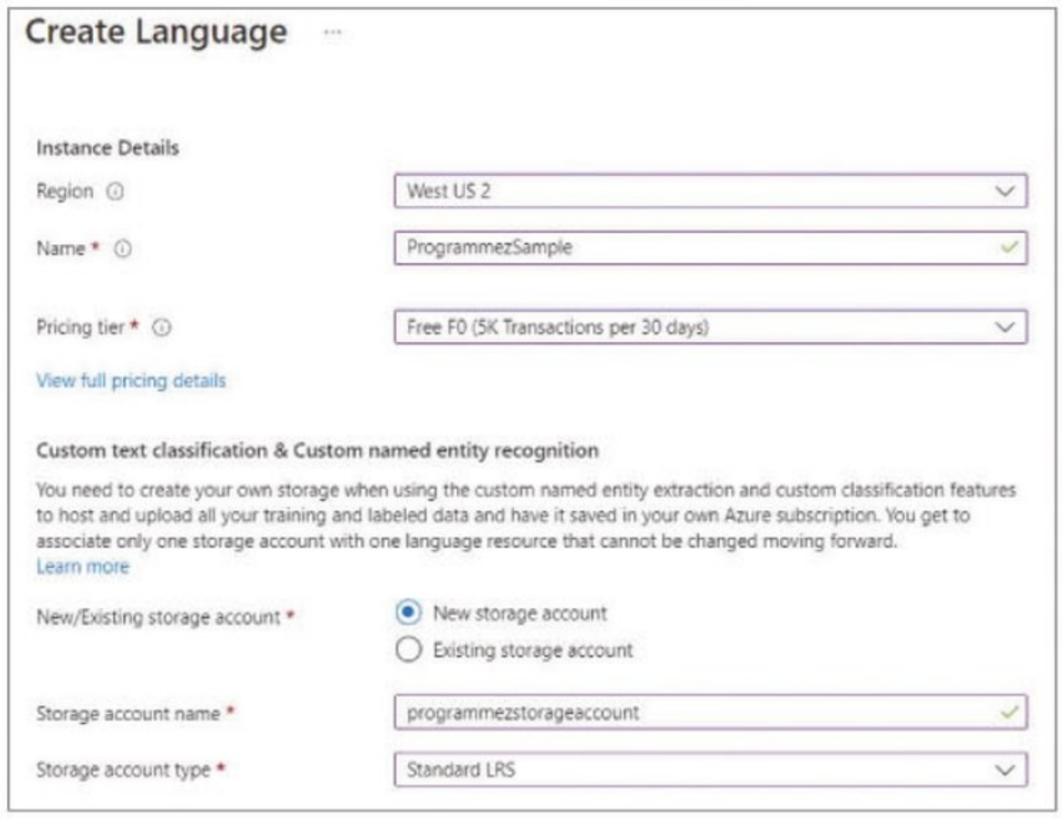


Figure 5

Text Classification va donc récupérer en input (en entrée) un texte, par la suite Azure via vos modèles déduira son label (catégorie) et vous donnera soit via Langage Studio soit via l'api votre label (plus le texte sera long plus la déduction sera facile). **Figure 3**

Voici un schéma des étapes complètes qui devront être effectuées pour la mise en place de notre IA :

1. Nous devons définir un schéma (Define schema), autrement dit nous devons déduire les labels que nous voulons que notre IA déduise
2. Par la suite nous allons ajouter des données dites d'entraînement (Tag data). Ces données seront donc tagger par nos labels et Azure s'en servira pour créer un modèle baser sur ces données d'entraînement. Encore une fois plus vous avez de données tagger plus votre modèle seras performant.
3. L'étape 3 est la création du modèle (train model), ici nous n'avons rien à faire Azure s'en occupe pour nous.
4. Les autre étapes (View model evaluation, improve model et deploy model) sont uniquement à but technique à ce stade de l'article rien ne sert de si attarder.
5. La dernière étape quant à elle consiste à nous renvoyer du texte classifié (classify text), autrement à déduire le label d'un texte inconnu.

Vous verrez que toutes les étapes de mise en place de notre IA sont d'une simplicité enfantine. Merci Azure.

Étape 1

Vous devez vous connecter au portail Azure et sélectionner Azure Cognitive Service Language. Puis sélectionner Text classification comme l'exemple qui suit : **figure 4**

Ensuite, vous vous retrouvez sur une page classique de création de ressource sous Azure. Vous devrez donc sélectionner votre groupe de ressources (=les ressources que je vais provisionner et utiliser pour le projet) ainsi que la région (= où je déploie mon projet) et le nom de votre projet. De plus, vous devrez créer un compte de stockage associé à ce projet. Il faut simplement lui donner un nom et un type (dans notre exemple LRS Standard). **Figure 5**

Après quelques secondes vos ressources seront déployées et disponibles. Passons à l'étape 2.

Étape 2

Ajoutons maintenant les données d'entraînement (Tag data). Ces données seront donc tagger par nos labels. Dans le contexte de notre projet, je devrais avoir plusieurs fichiers représentant mes pièces jointes et les tagger par les labels (RH, IT, COM). Ici je vais utiliser les données pré-tagger de Microsoft. Je vais donc les stocker dans mon storage account. Si comme moi vous utilisez les fichiers d'exemple, vous verrez qu'il y a environ 210 fichiers txt contenant différents textes ainsi qu'un fichier JSON. Les fichiers txt représentent nos données d'entraînement (ces données sont cruciales pour entraîner les modèles), les fichiers JSON nous permettent de déclarer nos labels(catégories) ainsi que de tagger nos fichiers d'entraînement. Voici à quoi ressemble ce fichier :

```
{
  "projectFileVersion": "2022-05-01",
  "stringIndexType": "Utf16CodeUnit",
  "metadata": {
    "projectName": "WebOfScience",
    "storageInputContainerName": "example-data",
    "projectKind": "CustomSingleLabelClassification",
    "description": "Trying out single label text classification",
    "language": "en",
    "multilingual": false,
    "settings": {}
  },
  "assets": {
    "projectKind": "CustomSingleLabelClassification",
    "classes": [
      {
        "category": "Computer_science"
      },
      {
        "category": "Electrical_engineering"
      },
      {
```

```
    "category": "Psychology"
  },
  {
    "category": "Mechanical_engineering"
  },
  {
    "category": "Civil_engineering"
  },
  {
    "category": "Medical"
  },
  {
    "category": "Biochemistry"
  }
 ],
 "documents": [
  {
   "location": "109.txt",
   "language": "en-us",
   "class":
    {
     "category": "Mechanical_engineering"
    }
  },
  {
   "location": "095.txt",
   "language": "en-us",
   "class":
    {
     "category » : "Mechanical_engineering"
    }
   }
  ]
 }
}
```

La partie jaune représente la déclaration de nos différents labels et la partie bleue représente la liste des fichiers ainsi que le tag associé. Nous avons 7 labels (catégories). Nous devons les stocker dans notre compte de stockage. Pour ce faire nous devons simplement aller dans la partie ***containers*** et en créer un nouveau. À ce stade, n'oubliez pas de rendre votre container public. **Figure 6**

Nous en avons terminé avec le portail Azure et nous n'avons plus qu'à nous rendre sur le site suivant (Language Studio) https://language.cognitive.azure.com/home. Une fois sur ce portail vous devrez sélectionner le groupe de ressource où se trouve le projet d'Azure Cognitive Service. **Figure 7**

Il faut créer le projet de cognitive service de Text Classification et le lier à notre compte de stockage (storage account). À la dernière étape, vous devrez sélectionner le conteneur qui contient l'ensemble des fichiers tagger. Vous devrez aussi sélectionner le fichier JSON qui contient la liste des fichiers ainsi que leurs labels associés : **Figure 8**

Bravo ! Notre projet Text Classification est créé. Reste à lancer l'entraînement de notre modèle et nous aurons notre IA customisée.

Le portail Language Studio nous permet d'avoir un récapitulatif des fichiers d'entraînement ainsi qu'aux labels associés. **Figure 9**

Étape 3

Dans cette étape nous devons simplement lancer l'entraînement de notre modèle, pour cela il suffit de se rendre dans l'onglet ***training jobs*** et de cliquer sur ***start training jobs***. Il nous suffira de donner un nom à notre modèle et de définir la part des fichiers d'entraînements qui serviront et la part qui servira à son évaluation (par défaut les valeurs sont de 80% / 20%). **Figure 10**

Après quelques minutes notre modèle sera prêt et nous pourrons le déployer pour pouvoir l'utiliser. Pour cela nous devons simplement nous rendre sur l'onglet ***Deploying model*** et cliquer sur ***Add deployment***. Nous devrons donner un nom à notre déploiement et lui associer notre modèle crée à l'étape précédente. **Figure 11**

Le déploiement terminé nous pourrons utiliser notre IA. Pour rappel, nous utilisons le sample de Microsoft via l'outil ***Microsoft Language Studio***. Cependant lors du projet client

Figure 7

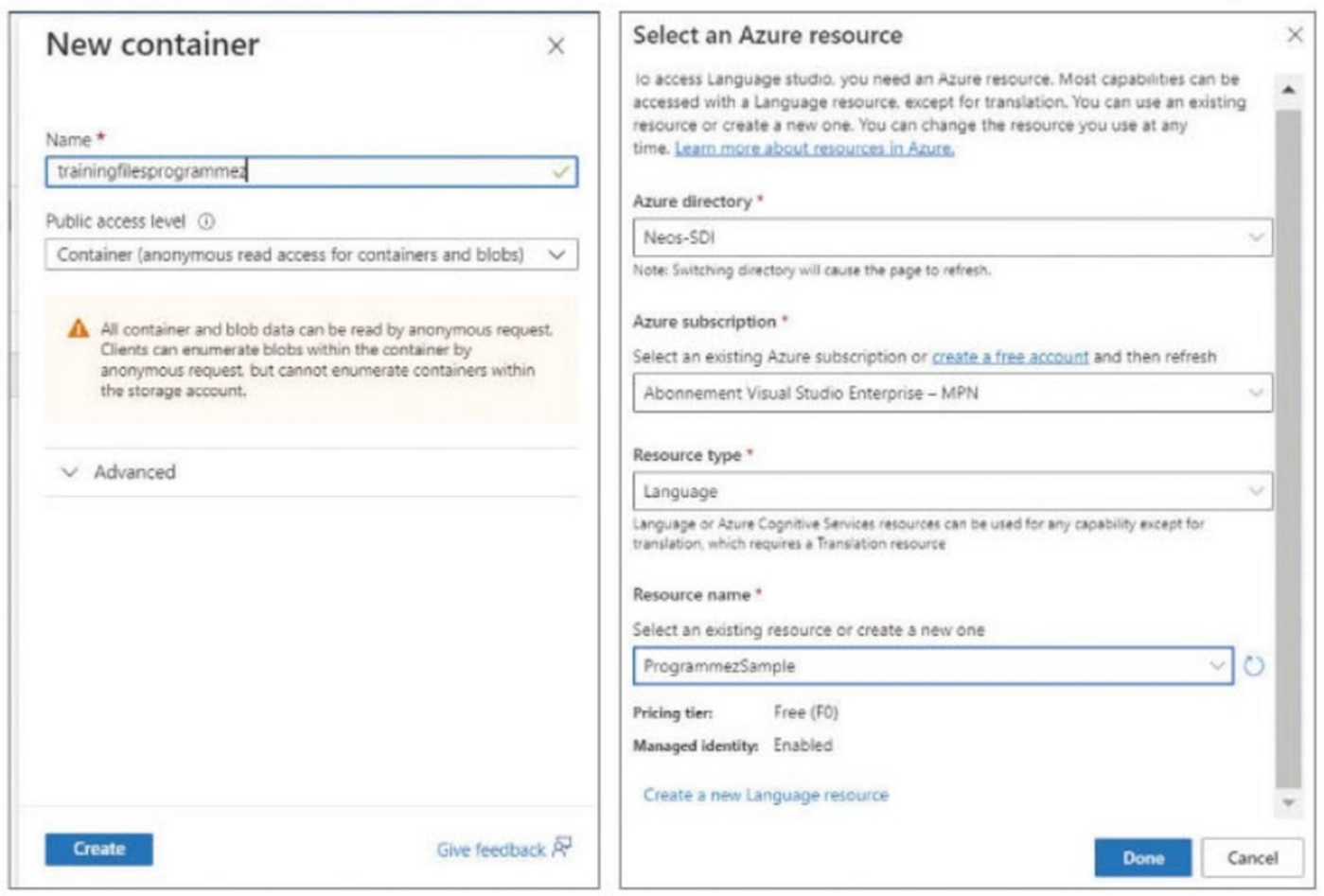


Figure 6

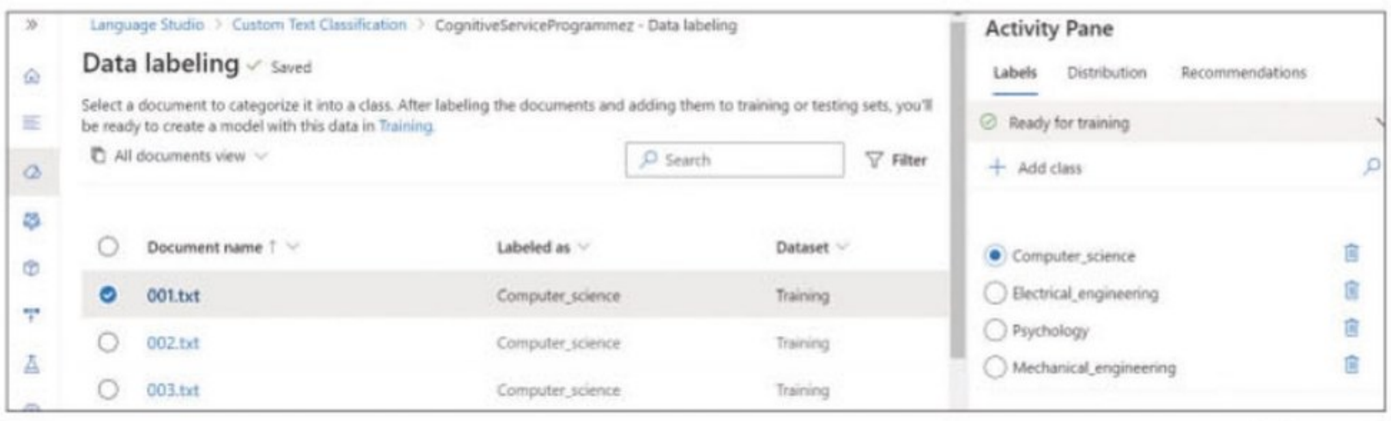


Figure 9

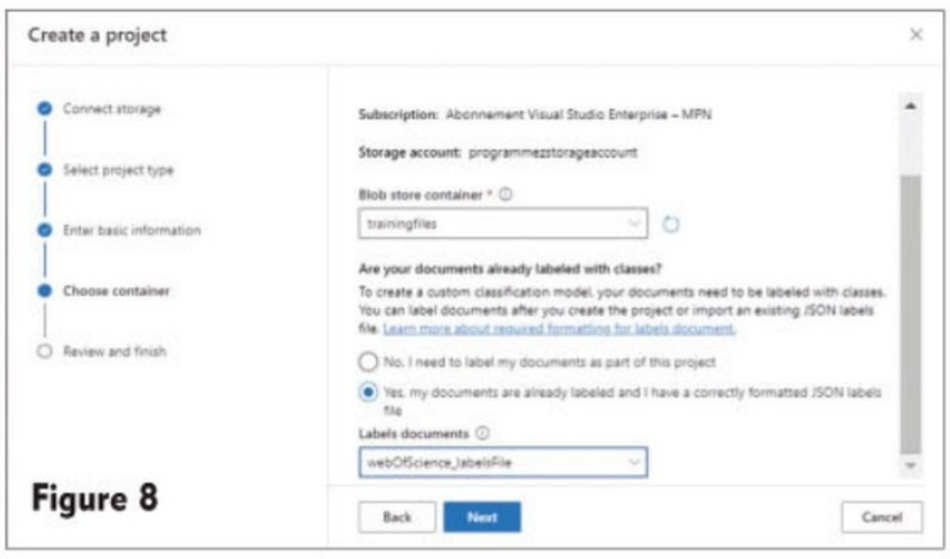


Figure 8

j'avais utilisé l'API liée à mon déploiement. Ici nous allons uniquement utiliser ***Language Studio***, pour ce faire il nous suffit de nous rendre dans l'onglet ***Testing Deployement*** puis de simplement coller un texte de test. Ici je choisis d'utiliser le texte d'un fichier d'entraînement. **Figure 12**
Après avoir cliqué sur le bouton Run the test, vous verrez la déduction de l'IA s'afficher juste en dessous du texte. Nous retrouvons plusieurs informations sous forme de JSON.

```
{
  "classes": [
    {
      "category": "Computer_science",
      "confidenceScore": 0.92
    }
  ]
}
```

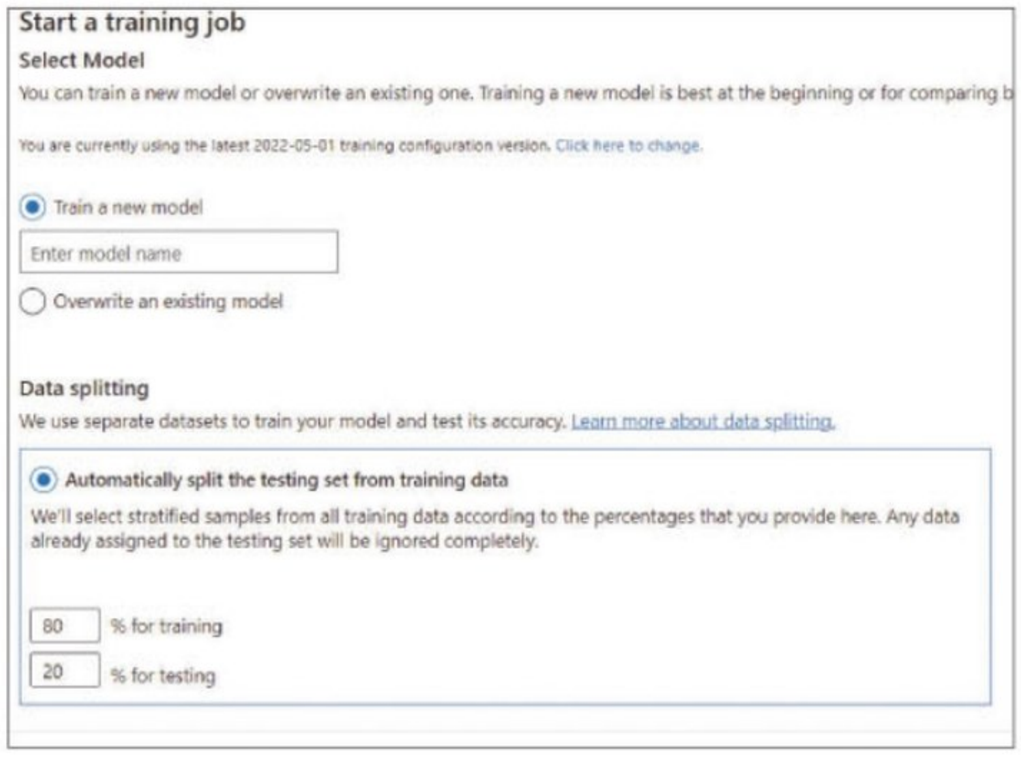


Figure 10

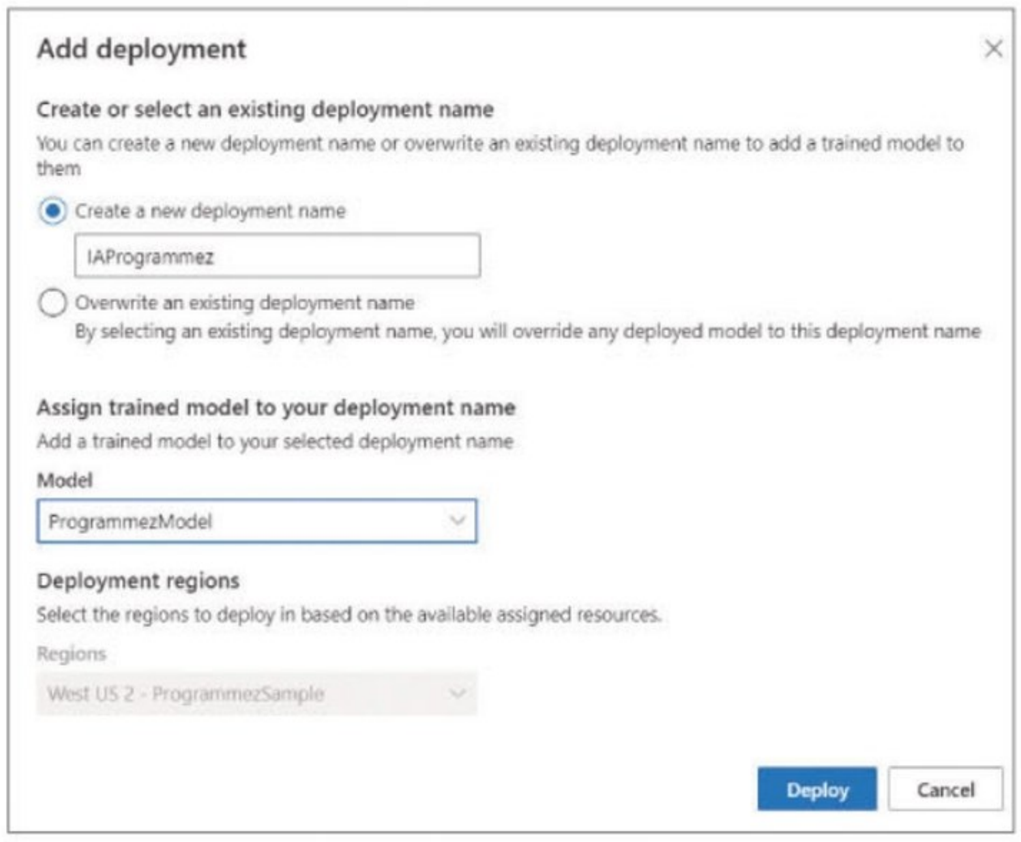


Figure 11

Tout d'abord nous aurons la catégorie déduite ici Computer_science puis nous retrouverons le pourcentage de sureté de la prédiction.
Notre modèle est plutôt précis. Azure rajoute plusieurs petites fonctionnalités intéressantes comme les performances de notre modèle. Pour voir nos performances, il suffit de se rendre sur l'onglet model performance. Sur la page de base, nous retrouverons notre score de précision sur l'ensemble des fichiers d'évaluations évoqué plus haut. **Figure 13**

Conclusion

Dans cet article nous avons pu voir comment créer une IA. Azure Cognitive Services m'a permis de répondre à la demande du client, et ce, sans jamais avoir eu besoin de savoir comment créer une IA. Cependant après une amélioration du projet, certaines limites sont apparues.
La première, et la plus évidente, est le prix des services. Avec les clouds publics, nous utilisons des services à la demande et à chaque service, ou presque, correspond une tarification plus ou moins élevée.
La seconde limite est en partie due au manque de fonctionnalité en ce qui concerne l'amélioration de notre modèle. À ce jour, l'API d'Azure ne permet pas de rajouter des fichiers d'entraînement à notre modèle et de le réentraîner.
Pour répondre à notre question évoquée plus haut, est ce que l'avenir de l'IA se trouve dans le cloud pour nous développeurs ? Pas encore. Actuellement pour des tâches simples, oui Azure Cognitive Service convient parfaitement. Mais quand est-il des IA plus complexes ? Eh bien nous ne pouvons tous simplement pas encore les créer sauf à passer par de la programmation bas niveau et des ressources spécifiques.

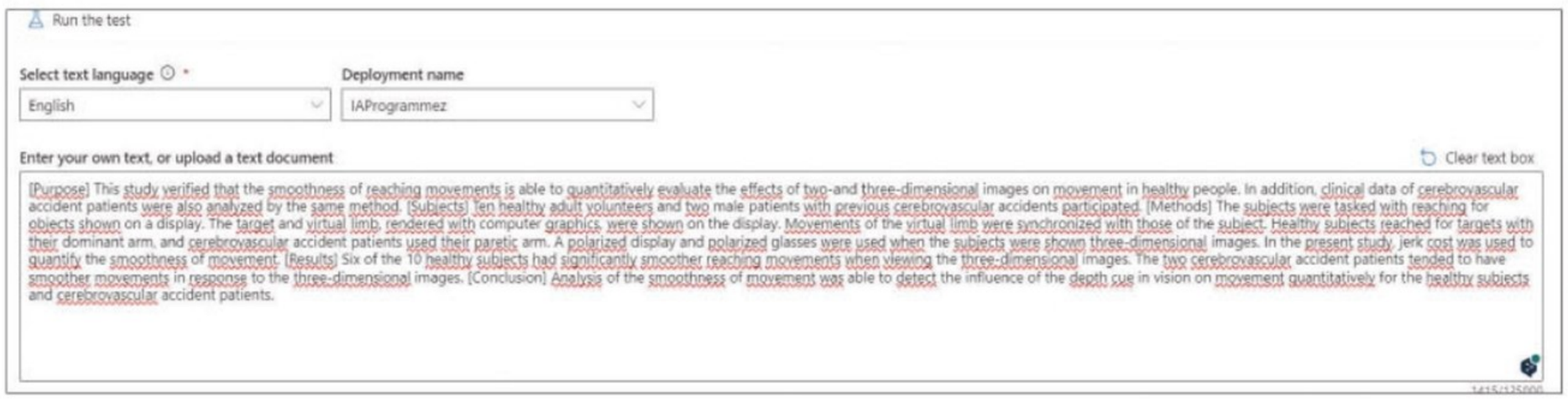


Figure 12

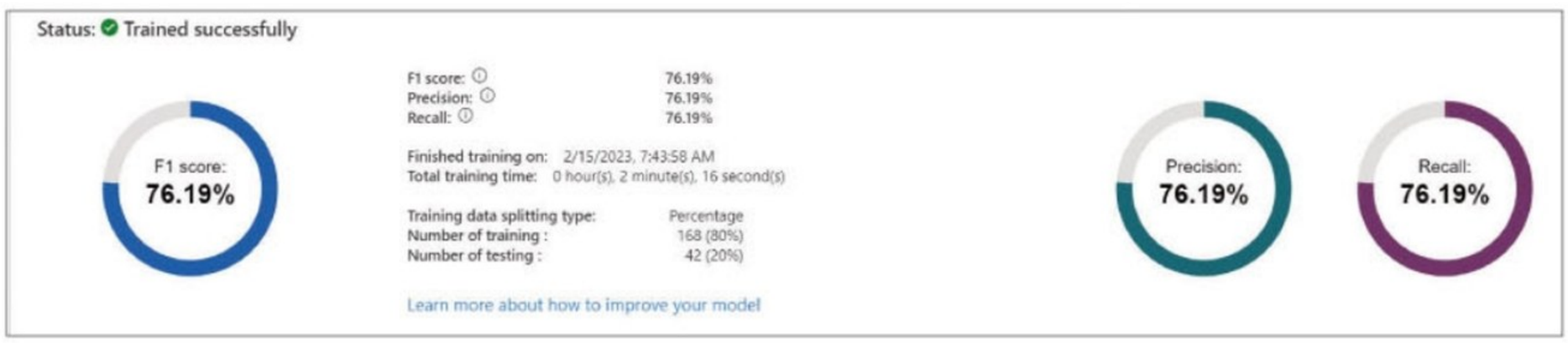


Figure 13

Serverless : comment en tirer parti ?

Beaucoup situent le début du cloud computing au milieu des années 90, lorsque les entreprises ont pu pour la première fois louer des serveurs dédiés sur Internet. Cependant, une grande partie des bases du cloud computing a été posée dans les années 1960. À cette époque, les instituts de recherche et les universités ont commencé à expérimenter l'accès à distance à des machines partagées, et ont entamé le voyage vers le cloud tel que nous le connaissons aujourd'hui.



Simon Shillaker
Engineering Manager, Serverless Compute

Scaleway
The cloud of choice

Trois principes ont stimulé l'innovation dans le cloud : la simplicité d'utilisation, la mise à l'échelle et le coût. Aujourd'hui, ces trois principes s'incarnent dans les nouvelles technologies dites "Serverless", qui rendent la construction de systèmes distribués accessible à moindre coût.

La simplicité, l'évolutivité et le coût.

Simplicité. Dans les années 1960, tout comme aujourd'hui, gérer son propre matériel était difficile. Les mainframes étaient des machines énormes et complexes, dont la maintenance nécessitait de l'expertise et du temps. Bien que les serveurs d'aujourd'hui soient plus petits, plus simples et plus robustes, il est toujours difficile d'installer des racks et des switches, de brancher des câbles, de configurer des topologies de réseau et de remplacer des disques défaillants. Le cloud d'aujourd'hui évite aux utilisateurs ces tracas, tout comme le time-sharing l'a fait dans les années 60.

Évolutivité. Le time-sharing et le cloud moderne consistent tous deux à donner aux utilisateurs l'accès à un pool de ressources plus grand que celui auquel ils auraient autrement accès. Dans les années 60, cela signifiait connecter un petit terminal client à un seul mainframe ; aujourd'hui, cela signifie connecter n'importe quel appareil en réseau à des datacenters gigantesques, avec un pool fonctionnellement infini de calcul, de stockage et de réseau.

Coût. La partie "sharing" du time-sharing a permis d'accéder à des machines mainframe coûteuses sans avoir besoin d'en acheter une pour chaque utilisateur. Ce sont exactement les mêmes motivations qui animent le cloud aujourd'hui ; en mettant les ressources en commun, le coût du matériel est amorti entre les utilisateurs. En conséquence, aujourd'hui, les entreprises peuvent exploiter de grands systèmes distribués servant des centaines de milliers de clients, tout en payant une fraction du coût du matériel sous-jacent et de l'expertise associée.

Comment sommes-nous passés de la multipropriété au cloud moderne ?

La transition du time-sharing au serverless a été rendue possible par plusieurs changements clés dans le monde de l'informatique : l'essor des "wide area networks" (WAN), le matériel de moins en moins cher et le développement de la virtualisation. Grâce à ces outils, nous avons pu passer du partage de grandes machines uniques au sein d'une organisation au partage de grands pools de machines via Internet.Les premiers fournisseurs de cloud sont apparus dans les années 90, offrant aux utilisateurs l'accès à des machines "bare metal". Chaque utilisateur avait le contrôle d'une ou plusieurs machines physiques pendant une période donnée. Comme cela coûtait cher, les utilisateurs exécutaient plusieurs applications sur la même machine physique. L'utilisation d'une seule grande machine pour exécuter plusieurs applications rendait plus difficile la gestion de la capacité, et la mise à niveau d'un seul hôte risquait de causer la défaillance de plusieurs applications.

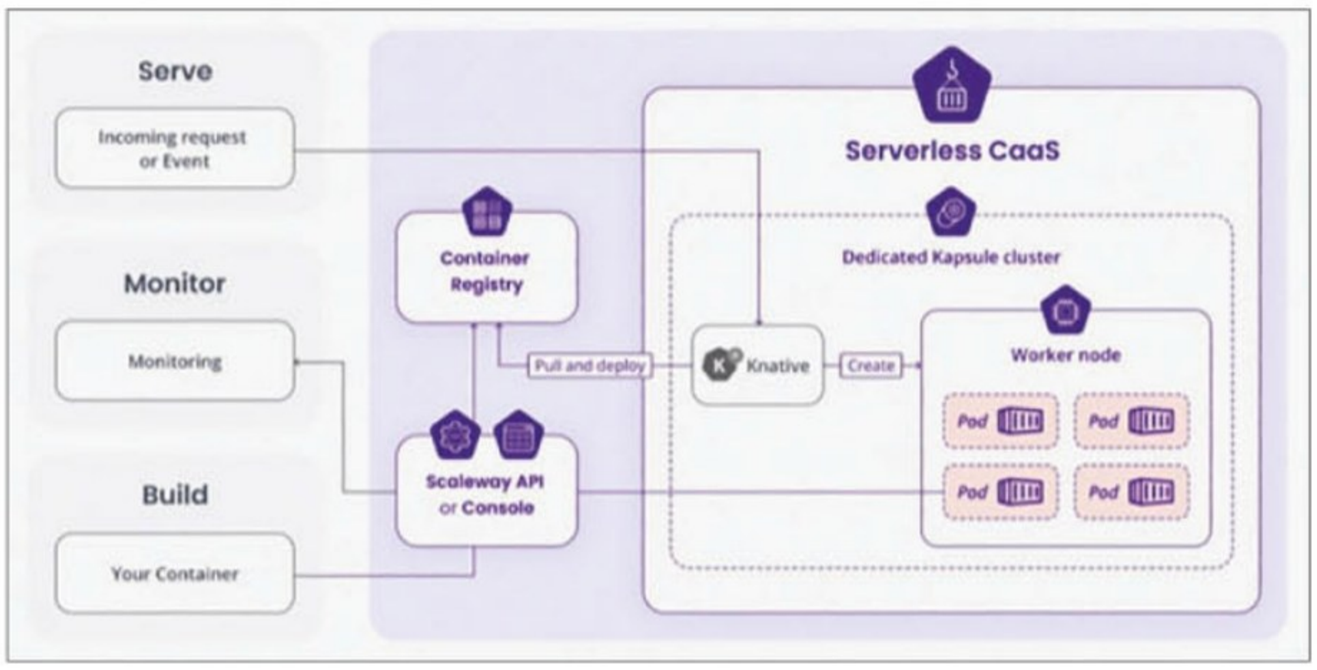
Séparer les grandes machines en petits morceaux a été rendu possible par l'arrivée de la virtualisation moderne. Bien que les idées derrière la virtualisation remontent aux années 1970, la virtualisation supportée par le matériel et les hyperviseurs modernes sont arrivés au début des années 2000, inaugurant l'ère du cloud computing basé sur les machines virtuelles. Les fournisseurs pouvaient désormais subdiviser en toute sécurité de grandes machines physiques en machines virtuelles (VM) plus petites, et les utilisateurs provisionner, faire évoluer et mettre à jour des VM à usage unique pour chacune de leurs applications. À mesure que les charges de travail des utilisateurs devenaient plus granulaires, les fournisseurs ont pu davantage utiliser leur infrastructure en regroupant plus étroitement les utilisateurs sur les hôtes physiques sous-jacents.

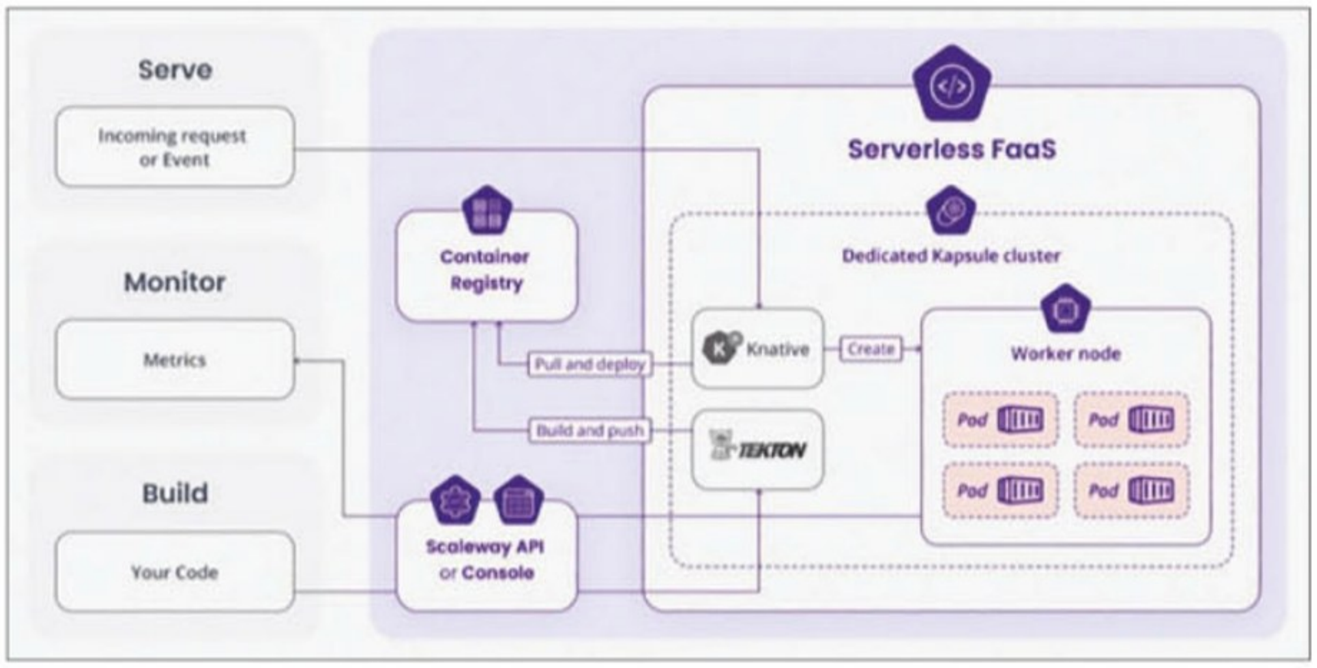
Docker et Kubernetes ont été lancés publiquement en 2013, lançant l'ère du cloud computing conteneurisé. Bien que les idées de virtualisation du système d'exploitation remontent à la fin des années 90 avec les jails FreeBSD, ce n'est qu'avec Docker que cette forme de virtualisation légère a été largement adoptée. Les conteneurs ont permis aux utilisateurs de diviser leurs applications en blocs encore plus petits avec une durée de vie plus courte, ce qui a conduit à une vision plus éphémère des ressources de calcul, illustrée par la célèbre citation de Bill Baker selon laquelle il faut traiter l'infrastructure comme "du bétail et non des animaux de compagnies" ("cattle, not pets").

Serverless : amener les principes du cloud à de nouveaux extrêmes

Serverless a vu le jour avec la sortie d'AWS Lambda en 2013. Bien que cela fasse presque 10 ans, le "serverless" n'en est encore qu'à ses débuts par rapport à d'autres modèles de cloud computing. Depuis 2013, Serverless est devenu un terme surchargé et mal défini. Pour certains, il ne s'agit encore que d'AWS Lambda et de Functions-as-a-Service (FaaS), mais pour beaucoup d'autres, c'est devenu une philosophie générale pour créer des produits cloud simples à utiliser et se mettant à l'échelle automatiquement (auto-scaling).

Pour expliquer ce qu'est Serverless, nous pouvons revenir aux trois mêmes principes de simplicité, de mise à l'échelle et de





coût. Serverless est l'incarnation la plus extrême de ces trois principes dans le cloud computing actuel, et vise à créer des systèmes incroyablement faciles à utiliser, infiniment scalable et véritablement facturés à la carte. Ces idées ne sont pas spécifiques à un seul produit et peuvent correspondre à des produits de calcul, de stockage et de mise en réseau.

Premièrement, Serverless vise à isoler complètement les utilisateurs de la configuration ou du provisionnement de l'infrastructure, poussant ainsi le principe de "simplicité" à l'extrême. Les utilisateurs n'ont pas besoin de spécifier un système d'exploitation, un type d'instance, des règles de mise en réseau ou de stockage. Au lieu de cela, ils n'ont qu'à écrire leur code, l'envoyer, et le fournisseur s'occupe du reste.

Deuxièmement, Serverless porte l'auto-scaling à un nouveau niveau, en passant de zéro à l'infini par défaut. Tous les produits Serverless passent automatiquement en veille (voir s'éteignent complètement) lorsqu'ils ne sont pas utilisés et démarrent, se mettent à l'échelle en fonction du besoin de ressources. Cet auto-scaling est rapide et granulaire, passant de zéro à des centaines de cœurs et de Go de mémoire en quelques secondes.

Enfin, la facturation Serverless offre un niveau de granularité et de rentabilité jamais atteint auparavant. La facturation Serverless à la carte est aussi précise que son auto-scaling. Les utilisateurs Serverless ne paient que ce qu'ils utilisent, le temps d'exécution en millisecondes, la RAM en Mo, et la bande passante entrante et sortante.

Les produits serverless les plus populaires aujourd'hui sont Functions-as-a-service (FaaS) et Containers-as-a-service (CaaS). Avec FaaS et CaaS, les utilisateurs divisent leur application en un certain nombre de petites fonctions ou conteneurs que le fournisseur met à l'échelle et exécute en réponse aux requêtes. De nombreux produits de stockage de données sont aussi serverless, mais pas toujours présentés comme tels. La majorité des solutions de stockage d'objets basées sur S3 disponibles aujourd'hui ne nécessitent qu'une configuration minimale, proposent une facturation à l'usage et gèrent automatiquement les ressources nécessaires.

Quand utiliser Serverless ?

En théorie, le choix d'utiliser serverless ne devrait poser aucun problème : moins de temps passé à configurer et à mettre à l'échelle signifie plus de temps passé à créer des applications ; l'auto-scaling signifie que le même code qui sert un utilisateur peut servir des milliers d'utilisateurs ; et la facturation à l'utilisation signifie qu'il n'y a plus à payer pour des ressources surprovisionnées ou sous-utilisées.

Cependant, comme Serverless n'en est qu'à ses débuts, il n'a pas l'écosystème de frameworks et de bibliothèques qui existe pour d'autres produits cloud. Il nécessite également un changement d'état d'esprit, car les développeurs s'habituent à l'idée de déléguer le contrôle de leur infrastructure aux fournisseurs de cloud. Pour donner un meilleur aperçu de l'état actuel de Serverless, nous allons aborder quelques cas d'utilisation où la technologie est bien adaptée, et quelques cas où elle l'est moins.

API Serverless. FaaS et CaaS sont des outils efficaces pour créer des API, où chaque endpoint d'API est présenté comme une seule fonction ou un seul conteneur. Plutôt que de devoir provisionner, configurer et mettre à l'échelle un équilibreur de charge et un ensemble de VM, l'utilisateur peut faire confiance au fournisseur Serverless pour gérer la mise à l'échelle, l'équilibrage de charge et la mise en réseau, ce qui lui permet de se concentrer sur la logique de son application. De plus, en écrivant chaque endpoint comme sa propre fonction/conteneur, ceux-ci peuvent se mettre à l'échelle indépendamment, ce qui libère le développeur du réglage de la logique d'auto-scaling. L'état de l'application derrière les API peut être stocké dans un stockage serverless, comme un stockage d'objets, ou une base de données NoSQL serverless.

Workflows serverless. Serverless peut être utilisé pour construire des workflows évolutifs, comme le traitement des données ou les pipelines de traitement d'images. En exprimant chaque étape du workflow comme une fonction/conteneur serverless et en les connectant via des files de messages, l'utilisateur n'a pas à déterminer le parallélisme et les niveaux de ressources appropriés pour chaque étape, ni à gaspiller des ressources sur des VM surprovisionnées.

La "glue" serverless. Les applications cloud distribuées ont souvent besoin de connecter des flux de données entre différentes parties de l'application. Ces données devront fréquemment être transformées ou agrégées, comme lors de la collecte d'événements d'une application Web pour les écrire dans un data warehouse. Plutôt que de gérer des ressources dédiées à cette tâche, les développeurs peuvent créer des fonctions serverless, déclenchées automatiquement par des événements en amont, pour s'exécuter à la demande et transformer ou agréger les données.

Quand ne pas utiliser Serverless ?

Certaines caractéristiques d'une application peuvent la rendre inadaptée à l'informatique serverless :

- **Haute performance.** Si votre application nécessite une puissance de calcul élevée, comme le rendu vidéo ou les simulations scientifiques, serverless peut ne pas être en mesure de fournir les ressources nécessaires.
- **Longue durée de vie.** L'informatique serverless est conçue pour les fonctions stateless et de courte durée. Si votre application nécessite des processus à longue durée de vie ou stateful, il est peut-être préférable d'utiliser un modèle traditionnel basé sur un serveur.
- **Strictement réglementé.** Serverless exécute votre application sur une infrastructure partagée, c'est-à-dire côte à côte avec d'autres applications potentiellement non fiables. Les fournisseurs de cloud font tout leur possible pour isoler les applications serverless, et il n'y a aucune raison de penser que le système est moins sûr que toute autre forme de cloud computing. Cependant, il est peu probable que cela réponde aux exigences réglementaires strictes autour des données sensibles, comme celles utilisées dans les applications médicales, gouvernementales ou d'application de la loi.
- **Trafic élevé.** Si votre application doit recevoir un volume élevé de trafic, le coût de son exécution sur une plateforme serverless pourrait être nettement plus élevé que l'utilisation d'un modèle traditionnel basé sur un serveur classique.
- **Très complexe.** Si votre application est complexe, il peut être plus difficile de la décomposer en petites fonctions indépendantes qui peuvent être exécutées sur une plateforme serverless.

Comment utiliser Serverless ?

Une fois que vous avez décidé de vous lancer dans votre voyage Serverless, il peut être difficile de savoir par où commencer. Tous les fournisseurs proposent une console basée sur un navigateur, où tu peux écrire du code, connecter des services et télécharger des dépendances. C'est génial pour expérimenter le serverless, mais cela peut être fastidieux et limitant lorsqu'il s'agit de gérer une application plus importante. Serverless Framework est un projet open source qui vise à résoudre une partie de la complexité de la gestion des applications serverless. Il adopte une approche déclarative, où tu peux écrire plusieurs fonctions serverless et définir la communication entre ces fonctions à l'aide de fichiers YAML. Serverless Framework fournit des back-ends pour tous les principaux fournisseurs de cloud, tu peux donc facilement migrer les fonctions entre eux.

Enfin, il existe une tendance émergente pour les frameworks Serverless de niveau supérieur, où le framework créera de manière transparente des ressources Serverless à partir de votre code. Ces frameworks adoptent souvent des conventions d'autres frameworks, par exemple, les annotations que l'on voit souvent dans les frameworks Web Python. Après avoir déployé une application serverless, analyser les métriques et journaux de logs est plus simple. Contrairement à d'autres systèmes distribués où cela peut être un travail un projet à part entière, la plupart des fournisseurs serverless proposent des produits d'observabilité prêts à l'emploi.

Comment Serverless va s'améliorer ?

Au cours des 5 prochaines années environ, nous pouvons nous attendre à ce que serverless s'améliore sur de nombreux points par rapport à ses faiblesses actuelles :

1. **Modèles de programmation.** Serverless est actuellement basé autour des fonctions, avec d'autres couches d'abstractions disponibles pour construire des workflows plus importants et des applications parallèles. La disponibilité de nouveaux frameworks serverless parallèles et event-driven facilitera l'écriture d'applications plus complexes.
2. **Stockage.** Le stockage serverless est encore difficile à gérer, car il doit être provisionné et mis à l'échelle indépendamment. Ce problème de scaling n'est pas en accord avec le principe de "simplicité" du serverless, et nous verrons probablement le calcul et le stockage serverless davantage associés à l'avenir.
3. **Mise en réseau et interconnexions.** Il est aujourd'hui difficile de connecter les différentes parties des applications serverless, car il est compliqué de connaître à l'avance la topologie et même le nombre de connexions nécessaires. Cela a été atténué par l'utilisation de produits de messagerie et de mise en file d'attente, mais cela ne prend pas en charge l'étendue des modèles et protocoles de communication offerts par un réseau standard.
4. **Prise en charge de différents langages.** Les cas d'utilisation et les modèles de programmation serverless d'aujourd'hui sont bien adaptés aux langages dynamiques tels que JavaScript et Python. Nous pouvons nous attendre à une meilleure prise en charge et à davantage de frameworks dans des langages compilés tels que Rust, Go et C++.
5. **Visibilité et prévisibilité des coûts.**
6. **Performances et ressources.** Les fonctions serverless n'offrent actuellement pas suffisamment de ressources, de modèles de communication ou de temps d'exécution pour lancer des tâches plus importantes et à haute performance, comme l'entraînement de modèles d'apprentissage. Cependant, la facilité, l'échelle et les avantages en termes de coûts du serverless sont toujours attrayants pour les développeurs d'applications à haute performance. Sans aucun doute le futur nous réserve beaucoup d'évolutions dans ce domaine.

Conclusion

Nous avons vu que, bien que serverless soit la pointe actuelle du cloud computing, il est construit sur les principes anciens de plusieurs décennies de la facilité, d'évolutivité et du coût. Il représente un point culminant de l'innovation et du développement qui nous a fait passer du temps partagé dans les années 1960 au pool de ressources hautement évolutif et fonctionnellement infini que sont les systèmes serverless d'aujourd'hui. Bien que le serverless n'en soit qu'à ses débuts, il existe déjà plusieurs cas d'utilisation convaincants, et il continuera à devenir plus facile, plus évolutif et moins cher à utiliser.

En savoir plus sur l'offre Serverless de Scaleway :
Containers : https://www.scaleway.com/en/serverless-containers/
Functions : https://www.scaleway.com/en/serverless-functions/



Sébastien Stormacq
Developer Advocate
Amazon Web Services

Les fonctions serverless, c'est pour tout le monde !

Dans cet article, je vais vous montrer une nouvelle option pour déployer vos apps web qui tournent dans des conteneurs. Une option qui ne nécessite pas de machine virtuelle et vous n'aurez donc rien à gérer ; une option qui scale automatiquement selon la charge réelle et qui est très abordable en termes de coûts. Cette solution s'appelle AWS Lambda. Il n'y pas besoin de modifier vos applications, ni de connaître AWS Lambda pour y déployer vos applications.

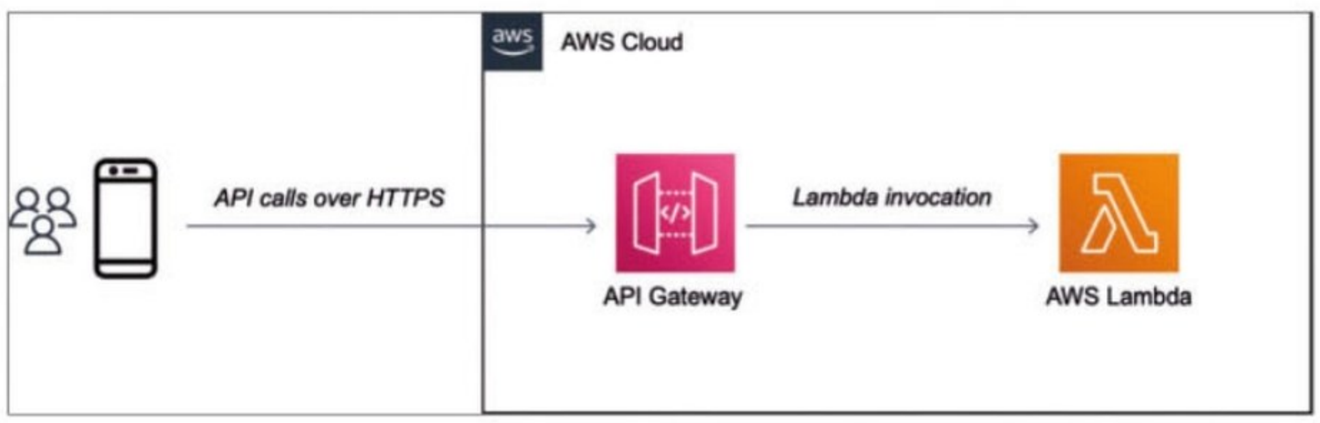
Grosso modo, il existe deux types d'applications web : les applications où le rendu HTML se fait dans le navigateur du client (développées avec React, Vue, ou Angular) et les applications où le rendu des views (HTML) ou des données (JSON) se fait côté serveur. Souvent, vos applications utilisent une combinaison des deux technologies : un rendu HTML dans le navigateur pour le front et une API en backend qui sert des données en JSON. De nombreux frameworks facilitent le développement web ou d'API sur le backend. Ils fournissent, ou travaillent derrière un serveur HTTP. Ils routent les appels HTTP vers votre code. Ils interprètent et donnent accès aux paramètres de la requête, aux en-têtes HTTP, etc. En retour, votre code collecte les données, applique un template pour les intégrer dans de l'HTML ou sérialise vos structures de données en JSON et retourne le tout à l'appelant.

Un service géré, facturé selon l'utilisation réelle

Souvent, vous déployez ce code dans des containers ou sur des machines virtuelles que vous payez 24/7. Vous devez aussi maintenir ces machines, par exemple installer les correctifs et mises à jour de l'OS, du langage et des frameworks et librairies que vous utilisez. Vous êtes aussi responsables de la disponibilité de votre solution : il faut mettre en place des mécanismes de surveillance et de redémarrage automatique. Enfin, vous êtes responsable de la scalabilité de votre déploiement : quand votre application a du succès, c'est à vous de provisionner (et gérer) plus de machines virtuelles, de containers ou les deux, et de balancer le trafic sur cette flotte. Pfew ! Quelle quantité de travail qui n'a rien à voir avec le développement de votre application.

D'un autre côté, vous avez entendu parler des fonctions serverless dans le cloud où on ne paye qu'à l'invocation et où le fournisseur de cloud gère la totalité des tâches opérationnelles décrites ci-avant. Mais vous pensez qu'il faut réécrire et repackager tout votre code pour que vous puissiez en tirer bénéfice. Rien n'est moins sûr !

Figure 1 : *Un déploiement typique de Lambda pour une API HTTP*



AWS Lambda, c'est quoi ?

AWS Lambda [https://docs.aws.amazon.com/lambda/latest/dg/lambda-foundation.html] est un service qui permet d'exécuter votre code dans le cloud, sans que vous n'ayez à gérer l'infrastructure sous-jacente. Quand vous développez une fonction Lambda, vous vous concentrez sur votre code et vous le déployez en un clic. Le service déploie votre code, se charge de la disponibilité, de la scalabilité, de la maintenance de l'OS sous-jacent, etc. Vous n'avez aucune tâche opérationnelle à vous occuper. Qu'il y ait un appel à votre application ou API par heure ou des dizaines de milliers, le service mettra à disposition la quantité d'instances nécessaires pour répondre à la demande. Vous ne payez que ce qui est utilisé réellement, calculé en millisecondes de temps CPU. S'il n'y a pas d'appels à votre application, vous ne payez rien.

Souvent, vous ne déployez pas une fonction Lambda toute seule. Soit vous attachez une URL à votre fonction Lambda, soit un autre service est responsable de l'invocation de votre fonction. Dans le cas d'une API, il s'agira probablement d'une API Gateway (qui agit comme point de terminaison HTTPS, package les arguments de la requête et les en-têtes et appelle la fonction Lambda). Mais beaucoup de services dans le cloud AWS peuvent déclencher des fonctions Lambda, ce qui en fait une brique de base de vos architectures orientées évènements. **Figure 1**

Le meilleur des deux mondes

Imaginez que vous puissiez déployer votre application telle quelle dans une fonction Lambda : plus de serveurs à gérer et une facturation selon l'usage réel. Cependant, il y a deux challenges pour déployer une application web dans une fonction Lambda.

Le premier est lié au modèle d'invocation de la fonction Lambda. L'environnement d'exécution d'une fonction appelle le service AWS Lambda pour obtenir le prochain évènement à traiter. Chaque évènement est passé à la fonction comme un paramètre JSON. C'est très différent d'une application web qui attend de recevoir des appels HTTP. **Figure 2**

Le second challenge est le packaging. Comment déployer votre application et ses dépendances ?

Le second défi est le plus facile à adresser, je commence par celui-ci. Il existe deux moyens de paqueter son code pour le déployer dans une fonction Lambda : les fichiers ZIP et les images conteneur, disponibles depuis décembre 2020. Donc, si votre application web est empaquetée dans un conteneur, vous pouvez la déployer dans une fonction Lambda. Et je parie que vos applications web sont déjà empaquetées dans

un conteneur. Mais comment appeler votre application web dans la fonction Lambda ? La solution vient des extensions Lambda. Une extension est un processus que vous déployez avec votre fonction Lambda. Il reçoit et peut agir sur les évènements reçus et les réponses faites par votre code. L'usage typique des extensions Lambda est l'observabilité ou la télémétrie.

L'idée est de combiner API Gateway et les extensions Lambda. Il faut écrire une extension Lambda qui reçoit des évènements HTTP envoyés par une API Gateway. Cette extension va recréer la requête HTTP initiale (le même verbe HTTP, les mêmes paramètres de la requête, les mêmes en-têtes HTTP, etc.) et appeler votre application web qui tourne dans son processus séparé. À son tour, votre application traite la requête et retourne des en-têtes HTTP (des cookies par exemple) et un body (HTML ou JSON). L'extension récupère l'ensemble de la réponse et la retourne à l'appelant, selon le format des réponses des fonctions Lambda.

Votre application continue d'exposer son endpoint HTTP et l'extension Lambda l'appelle sur http://127.0.0.1:8080 (ou n'importe quel port vous avez choisi d'utiliser). Dans cette architecture, le moteur d'exécution de Lambda est ignoré, les évènements sont interceptés par l'extension Lambda et traités par votre application qui n'y voit que du feu. La séquence est illustrée par la **figure 3**.

La bonne nouvelle c'est que vous ne devez pas écrire vous-même l'extension Lambda, vous pouvez utiliser l'extension open source écrite en Rust que nous fournissons [https://github.com/awslabs/aws-lambda-web-adapter].

L'autre bonne nouvelle, c'est que ce type de déploiement ne requiert aucun changement dans le code de votre application. Il faut juste ajouter une ligne à votre Dockerfile pour inclure l'extension Lambda dans votre conteneur.

En pratique

Je vais construire une API web et la déployer dans une fonction Lambda. Pour montrer que ça marche avec n'importe quel langage de programmation et framework web, je choisis un couple un peu moins populaire que ceux que j'ai cités dans mon introduction : Swift, le langage inventé et promu par Apple, et Vapor, le framework web open source pour Swift. Swift a l'avantage d'être fortement typé, concis et expressif, et facile à lire. Il y a tous les outils nécessaires et la communauté bienveillante et productive. C'est mon langage de choix pour coder sur le backend. Ce tutoriel fonctionne avec n'importe quel autre langage de programmation et framework web empaqueté dans un conteneur.

Après avoir installé Swift [https://www.swift.org/getting-started/] et Vapor [https://vapor.codes/] je commence par créer un nouveau project Vapor avec la commande `vapor new <directory>`. Je choisis de ne pas inclure Fluent, le framework ORM pour accéder à des bases de données, ni Leaf, le système de template pour générer de l'HTML, puisque je veux développer une API.

J'ouvre le projet dans VS Code et j'ajoute quelques lignes de code pour créer une API "echo", c'est-à-dire qu'elle va afficher en réponse l'ensemble des paramètres qui lui sont passés en entrée. (Si vous développez en Swift dans VS Code, n'oubliez pas d'installer le plugin Swift) **Figure 4**

Je teste l'application avec la commande `vapor run` et j'utilise la commande `curl` pour vérifier si tout marche bien. **Figure 5**

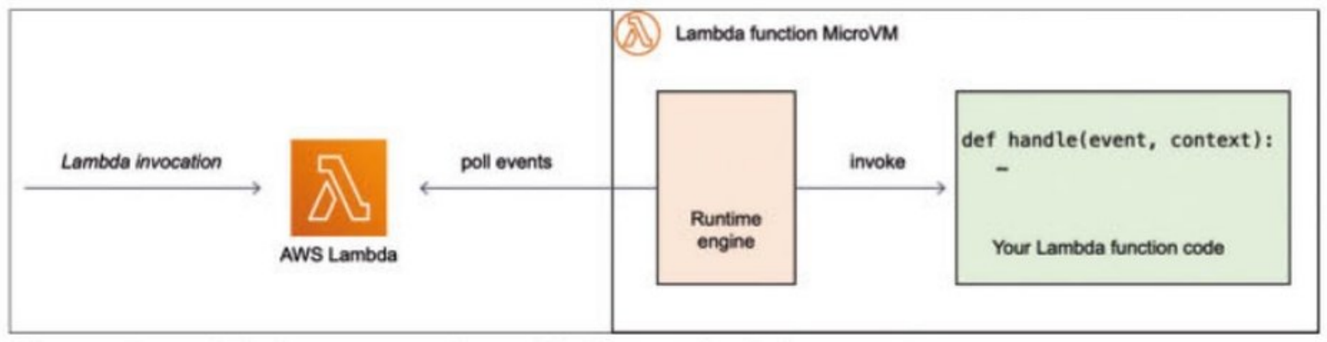


Figure 2 : *modèle d'invocation traditionnel des fonctions Lambda*

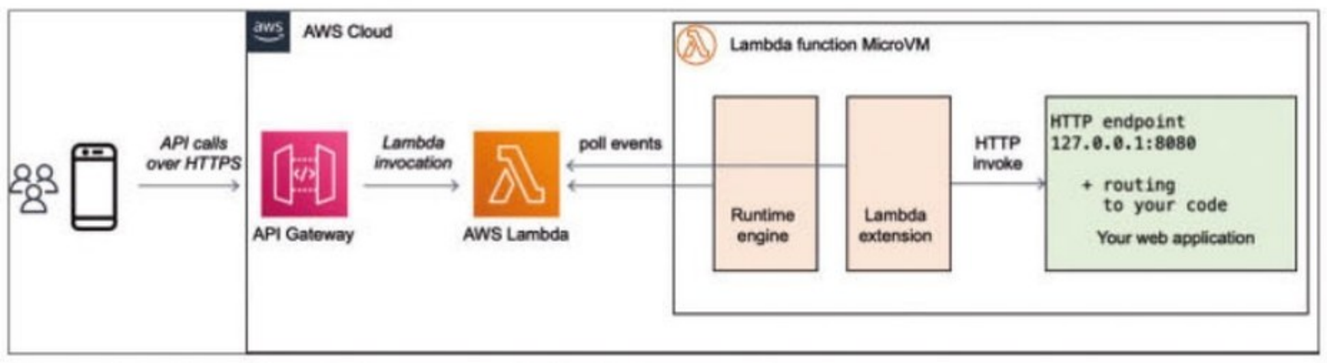


Figure 3 : *le flux d'appels via une extension Lambda*

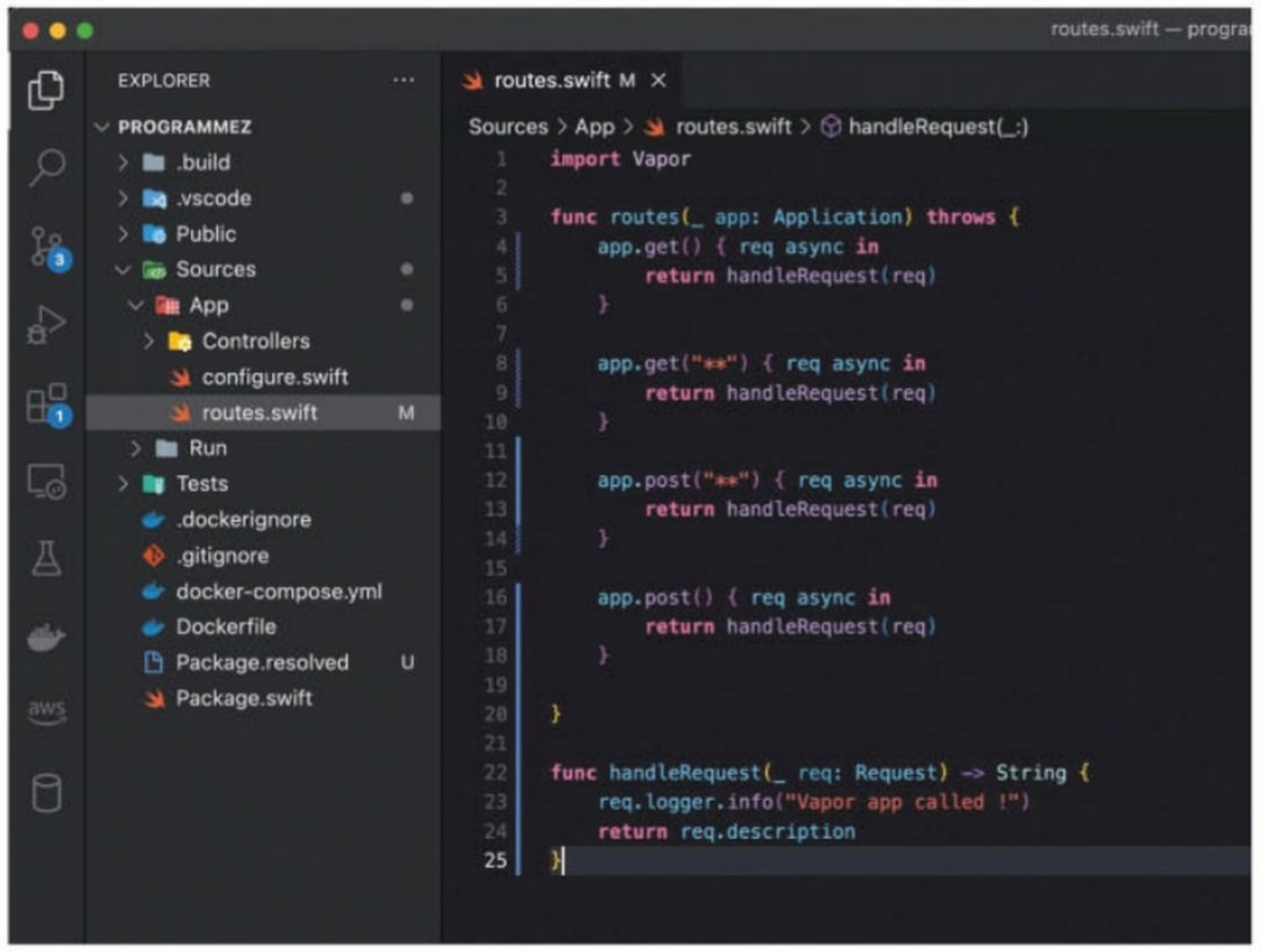


Figure 4 : *une API minimale en Vapor qui répond avec les paramètres reçus*

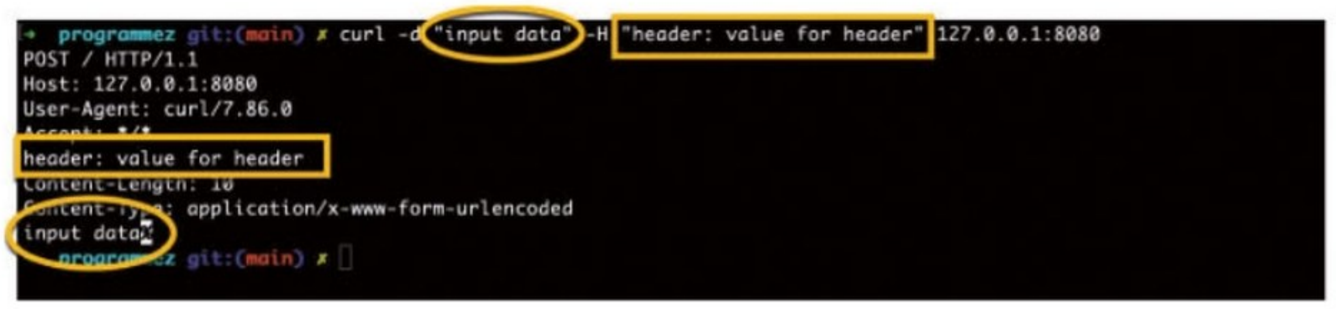


Figure 5

Vapor me fournit un Dockerfile par défaut. Donc je peux aussi tester l'app dans Docker. J'utilise les commandes `docker build` et `docker run`. Maintenant que je sais que mon conteneur fonctionne, je modifie le Dockerfile pour y ajouter l'extension Lambda. J'ajoute à la ligne 59 la commande qui prend l'image de l'extension et la copie dans mon conteneur.

```
COPY --from=public.ecr.aws/awsguru/aws-lambda-adapter:0.6.1 /lambda-adapter /opt/extensions/lambda-adapter
```

Je fais aussi quelques changements pour ne pas créer d'utilisateur `vapor` et ne pas changer les permissions sur les fichiers. Ceci ne fonctionne pas dans une fonction Lambda. Enfin, je change le port de l'application pour utiliser 9090 pour éviter un conflit lorsque je testerai l'application en local dans la section suivante. Les lignes ainsi modifiées ont une marque bleue dans la figure 9. Une fois les modifications faites, j'as-

semble mon conteneur avec la commande docker build utilisée précédemment pour être certain que tout fonctionne.

```
docker build -t programmez:arm64 .
```

Figure 6

Si vous avez déjà utilisé le répertoire public d'images de AWS, il se pourrait que vous receviez l'erreur 403 Forbidden en essayant de charger l'image de l'extension Lambda. C'est probablement parce que vous avez un token d'authentification expiré dans votre configuration docker. Dans ce cas, utilisez la commande suivante avant de faire le build :

```
docker logout public.ecr.aws
```

L'étape suivante est de déployer ce code sur AWS Lambda. En fait, il faut un peu plus que juste déployer le code, il faut aussi créer une API Gateway et lier les deux ensemble. Tout cela pourrait se faire dans la console AWS, mais, en tant que développeur, je préfère utiliser du code pour décrire l'infrastructure que je souhaite et utiliser un outil qui va créer cette infrastructure pour moi. De cette façon, je peux répéter mes déploiements dans différents environnements (dev, starting, production) et je peux mettre la description de mon infrastructure dans mon outil de contrôle de version de source, avec le reste des sources de mon application.

Pour ce faire, j'utilise AWS SAM [https://aws.amazon.com/serverless/sam/]. Après avoir installé la ligne de commande SAM, je crée un fichier qui décrit mon déploiement. Je pars d'un fichier fourni par la documentation et je le modifie pour mon application. Regardons en détails ce fichier template.yaml.

Figure 7

Les lignes 1 et 2 sont standard pour tous les templates SAM. À partir de la ligne 4, je définis les ressources que je veux créer dans le cloud. La première ressource est une fonction Lambda qui s'appelle "Programmez" et dont la description commence sur la ligne 5 jusqu'à la ligne 23. Sur la ligne 6, le type indique à SAM que je veux créer une fonction Lambda.

La ligne 8 indique que la fonction Lambda doit tourner sur une architecture Arm64. Lambda permet aussi de tourner sur une architecture x64. Le choix de l'architecture doit correspondre à celle de la machine que j'utilise pour la commande docker build. Dans mon cas, je build sur un Macbook Pro M1, je déploie donc sur Arm64. Attention si vous utilisez des chaînes de CI/CD, assurez-vous que votre infrastructure de build corresponde à l'environnement d'exécution.

La ligne 9 indique que le package est une image de conteneur (et non un fichier ZIP). La ligne 10 indique la quantité de mémoire que je souhaite allouer à l'environnement d'exécution Lambda. Notez que la puissance CPU va de paire avec la quantité de mémoire. Au plus vous allouez de mémoire, au plus de cores CPU sont disponibles. La documentation de AWS Lambda vous renseignera sur les détails. Attention, ce paramètre a un impact sur le prix. À nouveau, la page de tarifs de AWS Lambda vous donnera les détails.

Les lignes 13-15 définissent une variable d'environnement qui indique à l'extension Lambda sur quel port se trouve votre application. Le port par default est TCP 8080. Comme indiqué précédemment, je choisis d'utiliser le port TCP 9090 pour pouvoir faire des tests localement dans l'étape suivante. Les lignes 16-18 indiquent que je souhaite créer une API gateway pour déclencher ma fonction Lambda. Comme je n'indique ni chemin ni commande HTTP, l'API Gateway agira en passe-plat : tout ce qui est reçu sera envoyé à la fonction Lambda.

Les lignes 20-23 sont spécifiques à mon conteneur : où se trouve le Dockerfile, comment s'appelle-t-il et quels tags utiliser ? Enfin, les lignes 25 et suivantes définissent les valeurs à afficher en sortie. Dans ce cas-ci, j'affiche l'URL de l'API Gateway. De cette façon je connaîtrais l'URL à utiliser pour tester mon application. Je peux maintenant construire mon conteneur pour le préparer au déploiement. J'utilise la commande sam build. Après quelques minutes, vous devriez voir le message Build Succeeded. **Figure 8**

Dockerfile > ...

```
# ================================
# Run image
# ================================
FROM ubuntu:jammy

# Make sure all system packages are up to date, and install only essential packages.
RUN export DEBIAN_FRONTEND=noninteractive DEBCONF_NONINTERACTIVE_SEEN=true \
    && apt-get -q update \
    && apt-get -q dist-upgrade -y \
    && apt-get -q install -y \
      ca-certificates \
      tzdata \
# If your app or its dependencies import FoundationNetworking, also install `libcurl4`.
      # libcurl4 \
# If your app or its dependencies import FoundationXML, also install `libxml2`.
      # libxml2 \
    && rm -r /var/lib/apt/lists/*
COPY --from=public.ecr.aws/awsguru/aws-lambda-adapter:0.6.1 /lambda-adapter /opt/extensions/lambda-adapter
# Create a vapor user and group with /app as its home directory
#RUN useradd --user-group --create-home --system --skel /dev/null --home-dir /app vapor

# Switch to the new home directory
WORKDIR /app

# Copy built executable and any staged resources from builder
COPY --from=build /staging /app

# Ensure all further commands run as the vapor user
#USER vapor:vapor

# Let Docker bind to port 9090
EXPOSE 9090

# Start the Vapor service when the image is run, default to listening on 8080 in production environment
ENTRYPOINT ["./Run"]
CMD ["serve", "--env", "production", "--hostname", "0.0.0.0", "--port", "9090"]
```

Figure 6 : *les modifications faites au Dockerfile de Vapor*

template.yaml > {} Resources > {} Programmez > Type

```
aws://sam.schema.json (sam.schema.json)
1  AWSTemplateFormatVersion: '2010-09-09'
2  Transform: AWS::Serverless-2016-10-31
3
4  Resources:
     AWS: Add Debug Configuration
5    Programmez:
6      Type: AWS::Serverless::Function
7      Properties:
8        Architectures: [arm64]
9        PackageType: Image
10       MemorySize: 512
11       # Port of the Vapor container. Do not use default (8080) to avoid conflicts with Lambda runtime emulator
12       # https://github.com/awslabs/aws-lambda-web-adapter/issues/125
13       Environment:
14         Variables:
15           PORT: 9090
16       Events:
17         Root:
18           Type: HttpApi
19
20     Metadata:
21       DockerTag: arm64
22       DockerContext: .
23       Dockerfile: Dockerfile
24
25 Outputs:
26   ApiGateway:
27     Description: "API Gateway endpoint URL for Prod stage for Vapor"
28     Value: !Sub "https://${ServerlessHttpApi}.execute-api.${AWS::Region}.${AWS::URLSuffix}/"
29
```

Figure 7 : *un template de déploiement SAM*

Tester en local

SAM me permet de tester mon conteneur en local avec une émulation de API Gateway et Lambda. Pour ce faire, je lance le conteneur avec la commande sam local start-api.

SAM démarre un serveur en local sur le port TCP 3000. Je peux le tester avec une commande curl similaire a celle utilisée précédemment.

Déployer et Tester dans le cloud

Maintenant que je suis sûr que mon application fonctionne, je peux la déployer dans le cloud avec la commande sam deploy —guided. **Notez qu'il est nécessaire d'avoir un compte AWS pour passer cette étape**. L'option —guided n'est nécessaire qu'au tout premier déploiement. Cette option indique à SAM de collecter quelques paramètres du déploiement qui seront sauvegardés et ré-appliqués automatiquement lors des prochains déploiements de ce projet. Pour la plupart des questions, j'accepte la réponse proposée par défaut. **Figure 9**

C'est un bon moment pour se dégourdir les jambes et prendre une boisson. Le déploiement crée l'infrastructure dans le cloud : la fonction Lambda, l'API Gateway mais aussi quelques rôles et permissions pour que tout fonctionne ensemble.

Après une ou deux minutes, vous devriez voir un message de succès et l'URL de l'API Gateway juste créée. Comme je l'ai fait précédemment, je teste le déploiement avec la commande curl. J'utilise l'URL de l'API Gateway retournée par SAM.

Le premier appel peut prendre quelques secondes, le temps de charger l'image du conteneur depuis le répertoire d'images et démarrer le container et l'extension Lambda. C'est le phénomène de démarrage à froid de Lambda.

Combien ça coûte ?

Nous avons fait tout ça pour deux raisons. La première est de ne plus devoir gérer d'infrastructure, ne plus s'occuper de la haute disponibilité, de la scalabilité, etc.

La seconde raison est de ne plus payer pour des serveurs 24/7 mais juste pour ce qui est utilisé. Faisons donc une petite estimation de coûts.

Imaginons que cette API soit appelée 10.000 fois par heure pendant les heures de bureau (10h / jour / 5 jours par semaine) et 100 fois / heure le reste du temps. Chaque invocation prend 50 ms de temps d'exécution.

Je déploie sur un conteneur avec 512 MB de mémoire sur la région de Paris.

10.000 (invocations) x 10 (heures) x 5 (jours) x 4 (semaines dans un mois) = 2.000.000 invocations / mois.

(100 (invocations) x 14 (heures) x 5 (jours)) + (100 (invocations) x 24 (heures) x 2 (jours de week-end)) x 4 (semaines dans un mois) = 47.200 invocations / mois.

Soit un total de (2.000.000 + 47.200) * 50 ms * 0,001 (ms vers secondes) = 102.360 secondes de calcul.

102.360 secondes * 0,5 GB (mémoire) = 51.180 GB / secondes de temps CPU par mois, soit en dessous du tier de fonctionnement gratuit (400.000 GB / Sec par mois). Total : **$0.0 / mois**

Les invocations de la fonction Lambda sont facturées $0.20 par million, avec le premier million gratuit tous les mois.

Soit (2.047.200 - 1.000.000 (gratuit par mois) / 1.000.000) * $0.20 = $0.21

Selon les hypothèses énoncées ci-dessus, cette fonction me coûtera donc $0.0 + $0.21 = **$0.21 / mois**

L'API Gateway quant à elle est facturée $1.17 par million d'appels.

(2.047.200 / 1.000.000) * $1.17 = **$2.40 / mois**

```
Build Succeeded

Built Artifacts  : .aws-sam/build
Built Template   : .aws-sam/build/template.yaml

Commands you can use next
=========================
[*] Validate SAM template: sam validate
[*] Invoke Function: sam local invoke
[*] Test Function in the Cloud: sam sync --stack-name {{stack-name}} --watch
[*] Deploy: sam deploy --guided
→  programmez git:(main) ✗
```

Figure 8 : *SAM build est réussi*

```
→  programmez git:(main) ✗ sam deploy --guided

Configuring SAM deploy
======================

        Looking for config file [samconfig.toml] :  Not found

        Setting default arguments for 'sam deploy'
        =========================================
        Stack Name [sam-app]: programmez
        AWS Region [eu-central-1]:
        #Shows you resources changes to be deployed and require a 'Y' to initiate deploy
        Confirm changes before deploy [y/N]:
        #SAM needs permission to be able to create roles to connect to the resources in your template
        Allow SAM CLI IAM role creation [Y/n]:
        #Preserves the state of previously provisioned resources when an operation fails
        Disable rollback [y/N]:
        Programmez may not have authorization defined, Is this okay? [y/N]: y
        Save arguments to configuration file [Y/n]: y
        SAM configuration file [samconfig.toml]:
        SAM configuration environment [default]: dev

        Looking for resources needed for deployment:
         Managed S3 bucket: aws-sam-cli-managed-default-samclisourcebucket-z2rhh0echgwp
         A different default S3 bucket can be set in samconfig.toml
         Image repositories: Not found.
         #Managed repositories will be deleted when their functions are removed from the template and
 deployed
         Create managed ECR repositories for all functions? [Y/n]:
```

Figure 9 : *le tout premier déploiement avec SAM*

Le coût total de mon API sera donc de $2.61 / mois. Ce calcul ne tient pas compte du tier de fonctionnement gratuit valable les douze premiers mois de la vie d'un nouveau compte AWS. Ce calcul ne tient pas compte non plus des options de pré-payement ou d'engagement sur la durée qui permettent de réduire les coûts.

Vous pouvez consulter le détail du calcul et changer mes hypothèses en utilisant la calculatrice des tarifs AWS : https://bit.ly/3iZxPUC.

Essayez dès aujourd'hui

En modifiant quelques lignes de code dans votre fichier Dockerfile et en utilisant AWS SAM, vous avez empaqueté votre application web ou API existante dans une fonction AWS Lambda et vous pouvez la déployer sur une infrastructure serverless. Vous n'avez pas modifié le code de l'application elle-même.

Pour cet exemple, j'ai utilisé un combo moins populaire que les frameworks classiques que sont Spring, Django, Express ou https://dotnet.microsoft.com/en-us/apps/aspnet mais vous pouvez utiliser l'extension Lambda web-adapter avec n'importe quelle application web empaquetée dans un conteneur.

SAM permet beaucoup d'autres possibilités comme la gestion d'environnements multiples (dev, test, staging, prod par exemple) et le déploiement et l'intégration continue. Je vous laisse explorer ces possibilités en lisant la doc de SAM.

L'étape suivante est de protéger votre API par une des méthodes d'authentification supportées : OAuth2, AWS IAM, ou une authentification via une autre fonction Lambda. Ne déployez pas d'application sans authentification.

Rendez-vous sur la page de l'extension web de Lambda [https://github.com/awslabs/aws-lambda-web-adapter] et sur la page de SAM [https://aws.amazon.com/serverless/sam/].



Clément Sannier
Leader Technique MS

SQLI DIGITAL EXPERIENCE

Développer, déployer et sécuriser les accès aux ressources grâce à Azure AD

Azure Active Directory (AAD) est un outil précieux pour garantir la sécurité d'un service SaaS hébergée dans Azure. Au-delà d'offrir une gestion centralisée des comptes utilisateurs, il permet de sécuriser les communications entre les ressources sans exposer des informations sensibles. En suivant les étapes de ce guide, vous pourrez développer, déployer et sécuriser efficacement vos ressources avec Azure AD. À vos marques, prêt, sécurisez !

Première étape : développer une application

Nous optons ici pour l'application Blazor WebAssembly, qui utilise Blazor Wasm au niveau front et une API .NET au niveau back. La sécurité des échanges sera assurée grâce à l'utilisation d'un token délivré par Azure AD en utilisant le protocole OpenID Connect dérivé d'OAuth 2.0.
Pour mettre en place cette sécurité, il est nécessaire d'ajouter dans Azure AD les entrées appropriées. Cela implique d'inscrire deux applications : une application serveur et une application client, et de les interconnecter pour spécifier que le client est autorisé à utiliser le serveur. Vous pouvez suivre la documentation de Microsoft pour implémenter ces applications dans Azure AD : https://learn.microsoft.com/fr-fr/aspnet/core/blazor/security/webassembly/hosted-with-azure-active-directory?view=aspnetcore-7.0
Une fois la procédure terminée, cette commande générera le projet avec les informations adéquates :

```
dotnet new blazorwasm -au SingleOrg —api-client-id "{SERVER API APP CLIENT ID}" —app-id-uri "{SERVER API APP ID URI}" —client-id "{CLIENT APP CLIENT ID}" —default-scope "{DEFAULT SCOPE}" —domain "{TENANT DOMAIN}" -ho -o {APP NAME} —tenant-id "{TENANT ID}"
```

Après avoir lancé cette nouvelle application, vous serez redirigé vers la page de connexion Azure AD pour entrer vos identifiants.
En parcourant l'application template de Blazor Wasm, vous trouverez une liste dans la page « fetch data », alimentée par un appel à l'API back dont les données sont stockées de manière statique dans l'application. Il est ainsi conseillé d'utiliser une base de données. Nous utiliserons ici une base Azure CosmosDb (Base NoSQL proposée par Azure : https://learn.microsoft.com/en-us/azure/cosmos-db/introduction).
Cependant, sachez que la sécurisation de la base de données s'effectue de la même manière, quelle que soit la base choisie (Cosmos, SQL Server ou encore PostgreSQL).
Afin de relier la base de données à votre application, il est essentiel d'installer le logiciel CosmosDb Emulator et d'ajouter, dans notre solution, le package Nuget approprié :

```
dotnet add package Microsoft.Azure.Cosmos
```

Pour instancier la connexion avec CosmosDB, vous devrez vous référer aux informations de connexion accessibles sur la page d'accueil au lancement de l'émulateur : https://localhost:8081/_explorer/index.html
Il est possible d'utiliser la chaîne de connexion, mais elle ne sera pas nécessaire pour les prochaines étapes. Il est préférable d'utiliser un modèle à inclure dans le fichier « appsettings.json » :

```
"CosmosDb": {
  "Account": "https://localhost:8081/",
  "Key": "C2y6yDjf5/R+ob0N8A7Cgv30VRDJIWEHLM+4QDU5DE2nQ9nDuVTqobD4b8mGGyPMbIZnqyMsEcaGQy67XIw/Jw==",
  "DatabaseName": "",
  "ContainerName": "",
  "PartitionKey": "/id"
}
```

Bon à savoir : la clé est universelle et reste inchangée, quel que soit l'ordinateur sur lequel vous installez l'émulateur.
Pour une gestion plus efficace, vous pouvez créer un modèle C# qui va effectuer automatiquement un « binding » entre votre configuration et l'application lors de son démarrage, en utilisant le pattern "Options".:

```
public class CosmosDbConfigurationOptions
{
  public string Account { get; set; }
  public string Key { get; set; }
  public string DatabaseName { get; set; }
  public string ContainerName { get; set; }
  public string PartitionKey { get; set; }
}
```

Dans votre fichier « program.cs » vous pouvez ajouter les informations suivantes :

```
builder.Services.Configure<CosmosDbConfigurationOptions>(builder.Configuration.GetSection("CosmosDb"));
```

Ainsi, dans le constructeur des méthodes et grâce à l'injection de dépendance, vous récupérerez les informations de connexion par l'intermédiaire de « IOptions<T> » :

```
public WeatherForecastController(ILogger<WeatherForecastController> logger,
IOptions<CosmosDbConfigurationOptions> cosmosDbConfigurationOptions)
 {
    _logger = logger;
    _cosmosDbConfigurationOptions = cosmosDbConfigurationOptions;
 }
```

Vous pouvez à présent construire le client Cosmos et récupérer le container cible au sein duquel sont stockées les données :

```
var (account, key, databaseName, containerName, partitionKey) = _cosmosDb
ConfigurationOptions.Value;

using var cosmosClient = new CosmosClient(account,
        key);

var database = await cosmosClient.CreateDatabaseIfNotExistsAsync(databaseName);

await database.Database.CreateContainerIfNotExistsAsync(containerName,partitionKey);

var container = cosmosClient.GetContainer(databaseName, containerName);
```

Réalisez désormais une requête à la base de données et extraire le résultat :

```
var query = container.GetItemQueryIterator<WeatherForecast>();
var result = new List<WeatherForecast>();
while (query.HasMoreResults)
{
var current = await query.ReadNextAsync();
result.AddRange(current.Resource);
}
```

Pour une application destinée à être utilisée en production, je vous recommande de suivre les bonnes pratiques de la "Clean Architecture" en créant un projet « repository ».
Après avoir démarré l'application, vous constaterez que la liste est vide : vous pouvez à présent ajouter vos données.
Une fois l'opération réalisée, nous pouvons procéder à l'étape suivante : permettre l'accès à votre application de manière sécurisée, avec la création de ressources sur Azure !

Étape 2 : Création des ressources Azure

Plusieurs méthodes s'offrent à vous : vous avez la possibilité de créer les ressources manuellement à l'aide du portail, d'utiliser les templates ARM ou d'opter pour un outil de gestion de l'infrastructure en code tel que Terraform ou Pulumi.
Pour notre démonstration, nous utiliserons le "cloud shell" directement dans le portail Azure. **Figure 1**
Nous aurons besoin d'un groupe de ressources, d'un service d'application Web et d'une instance CosmosDb. Pour créer le groupe de ressources, vous devrez taper la commande suivante :

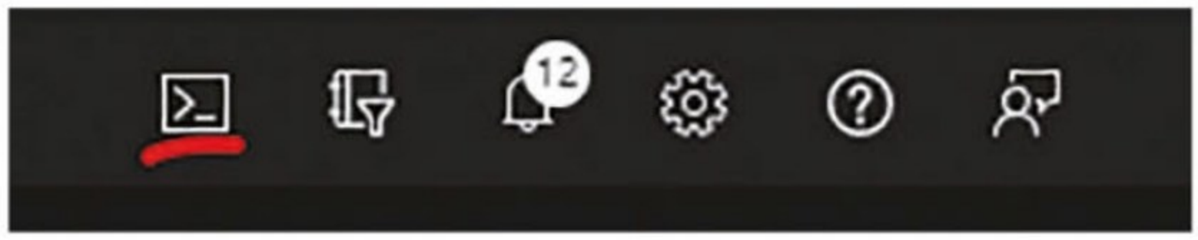


figure 1

```
az group create —name rg-programmez —location francecentral
```

Le groupe de ressource est ici situé en France, mais vous pouvez afficher les localisations valides à partir de la commande suivante :

```
az account list-locations —query "[*].name"
```

Pour créer le service d'application Web, vous aurez deux étapes à suivre : la première consiste à créer l'application plan, qui représente le serveur qui hébergera l'application Web :

```
az appservice plan create -g rg-programmez -n plan-programmez —is-linux
```

Dans cette commande, le -g indique le groupe de ressource cible. Il sera réutilisé à chaque création de ressources.
La deuxième étape consiste à créer l'application Web proprement dite :

```
az webapp create -g rg-programmez -p plan-programmez -n app-programmez-csa
—runtime "DOTNETCORE:7.0"
```

Vous pouvez accéder à la liste des différents runtime en utilisant la commande suivante :

```
az webapp list-runtimes
```

Enfin, pour créer une instance CosmosDb, tapez la commande suivante :

```
az cosmosdb create -g rg-programmez  -n cosmos-programmez-csa —enable-free-tier
```

Vous avez la possibilité de bénéficier d'une instance gratuite de CosmosDb avec chaque abonnement en utilisant l'argument -enable-free-tier. Pour créer une base de données complète, je vous recommande de créer directement la base de données :

```
az cosmosdb database create -n cosmos-programmez-csa  -g rg-programmez  —db-
name Programmez
```

Ainsi que le conteneur :

```
az cosmosdb sql container create -g rg-programmez -a cosmos-programmez-csa -d
Programmez -n Programmez-container -p "/id"
```

Toutes les ressources sont maintenant prêtes à être déployées et utilisées !
J'ai ajouté mon trigramme "csa" à chaque nom de ressource afin de le rendre unique. Vous pourrez ici le remplacer par le vôtre.

Pour en savoir plus sur les commandes, consultez ce lien : https://learn.microsoft.com/en-us/cli/azure/reference-index?view=azure-cli-latest.

Étape 3 : Le déploiement

Cette étape peut être réalisée avant ou après la sécurisation des accès, car elles sont indépendantes l'une de l'autre.
Pour déployer la solution, nous utiliserons la plateforme Github sur laquelle vous retrouverez le code.
Github Actions est une fonctionnalité qui permet de créer des workflows de CI/CD. Elle s'appuie sur la syntaxe YAML pour scripter les différentes commandes nécessaires pour compiler et déployer le code. Pour plus d'informations, n'hésitez pas à consulter ce lien : https://docs.github.com/fr/actions/using-workflows/workflow-syntax-for-github-actions.
Le fichier YAML se divise en deux tâches distinctes : la première étape, appelée "build", consiste à récupérer le code, restaurer les packages, compiler le programme et publier un artefact de la solution. La deuxième étape, appelée "deploy", récupère cet artefact et déploie l'application directement sur Azure. Le script de déploiement est le suivant :

```
name: Build and deploy Blazor app to an Azure Web App

env:
 AZURE_WEBAPP_NAME: app-programmez-csa   # set this to the name of your Azure Web App
 AZURE_WEBAPP_PACKAGE_PATH: '.'      # set this to the path to your web app project, defaults to the repository root
 DOTNET_VERSION: '7.0.x'              # set this to the .NET Core version to use
 PATH_SOLUTION : 'ProgrammezAzureSecurity'

on:
 push:
  branches: [ "main" ]
 workflow_dispatch:

permissions:
 contents: read

jobs:
 build:
  runs-on: ubuntu-latest
  steps:
   - uses: actions/checkout@v3

   - name: Set up .NET Core
    uses: actions/setup-dotnet@v2
   with:
    dotnet-version: ${{ env.DOTNET_VERSION }}

  - name: Set up dependency caching for faster builds
   uses: actions/cache@v3
   with:
    path: ~/.nuget/packages
    key: ${{ runner.os }}-nuget-${{ hashFiles('**/packages.lock.json') }}
    restore-keys: |
     ${{ runner.os }}-nuget-
  - name: Build with dotnet
   run: dotnet build ${{env.PATH_SOLUTION}}/*.sln —configuration Release

  - name: dotnet publish
   run: dotnet publish ${{env.PATH_SOLUTION}}/*.sln -c Release -o ${{env.DOTNET_ROOT}}/myapp

  - name: Upload artifact for deployment job
   uses: actions/upload-artifact@v3
   with:
    name: .net-app
    path: ${{env.DOTNET_ROOT}}/myapp

deploy:
 permissions:
  contents: none
 runs-on: ubuntu-latest
 needs: build
 environment:
  name: 'Development'
  url: ${{ steps.deploy-to-webapp.outputs.webapp-url }}

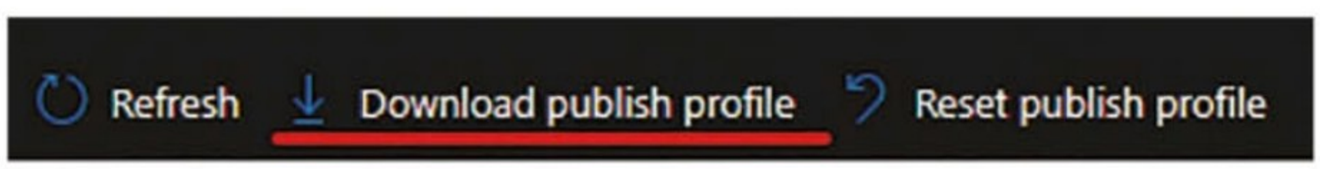
 steps:
  - name: Download artifact from build job
   uses: actions/download-artifact@v3
   with:
    name: .net-app

  - name: Deploy to Azure Web App
   id: deploy-to-webapp
   uses: azure/webapps-deploy@v2
   with:
    app-name: ${{ env.AZURE_WEBAPP_NAME }}
    publish-profile: ${{ secrets.AZURE_WEBAPP_PUBLISH_PROFILE }}
    package: ${{ env.AZURE_WEBAPP_PACKAGE_PATH }}
```

Le script de déploiement contient des variables d'environnement en début de script qui facilitent la transition d'un projet à un autre. Vous pouvez également remarquer à la fin du script la variable secrets.AZURE_WEBAPP_PUBLISH_PROFILE. Github Action attend le profil de publication dans la section "secrets". Ce profil se trouve sur la Web Application dans Azure : **Figure 2**
Il est nécessaire de copier / coller le contenu de ce profil dans un secret « Actions » de Github. Pour ce faire, rendez-vous dans les paramètres de votre projet : **Figure 3**

Attention, le nom de la variable doit être identique à celui défini dans la section « secrets », auquel cas Github ne sera pas en mesure de récupérer le profil de l'application.
Une fois que l'application est déployée, elle peut être lancée, mais n'a pas encore accès à la base de données. C'est le sujet de notre prochain chapitre.

Figure 2



Étape 4 : Sécurisation de l'accès aux ressources

Il existe plusieurs méthodes pour sécuriser l'accès à une ressource. Tout d'abord, vous pouvez sécuriser les données sensibles telles que les mots de passe en utilisant un coffre-fort. Azure Key-Vault de Microsoft Azure offre une solution de chiffrement et de sécurisation des informations sensibles très performante.

D'autre part, vous pouvez confier la sécurité des accès à Azure AD en utilisant des identités « managées ». Cela vous permet d'éviter de transmettre des informations de connexion. (https://learn.microsoft.com/fr-fr/azure/active-directory/managed-identities-azure-resources/overview) Pour cela, l'application Web doit nécessairement disposer d'une identité. Il existe deux façons de l'obtenir : en vous connectant au portail et en accédant à la Web App : **Figure 4**

ou en utilisant le cloud shell :

```
az webapp identity assign -g rg-programmez -n app-programmez-csa
```

Conservez précieusement votre ID pour les prochaines étapes.

Une fois que vous avez obtenu l'identité, vous pouvez assigner un rôle d'accès à l'application Web via son "principal ID". CosmosDB dispose de deux rôles préconfigurés : Read-only et Contributeur. Pour accorder le rôle Contributeur à la Web App, vous devez utiliser la commande :

```
az cosmosdb sql role assignment create —account-name cosmos-programmez-csa
—resource-group rg-programmez —role-definition-id 00000000-0000-0000-
0000-000000000002 —principal-id {service principal id}—scope "/"
```

L'identifiant par défaut pour le rôle contributeur est 00000000-0000-0000-0000-000000000002. Si vous voulez activer le rôle en lecture seule, l'identifiant correspondant est 00000000-0000-0000-0000-000000000001.

Pour vérifier que le rôle a été correctement assigné, vous pouvez utiliser la commande suivante :

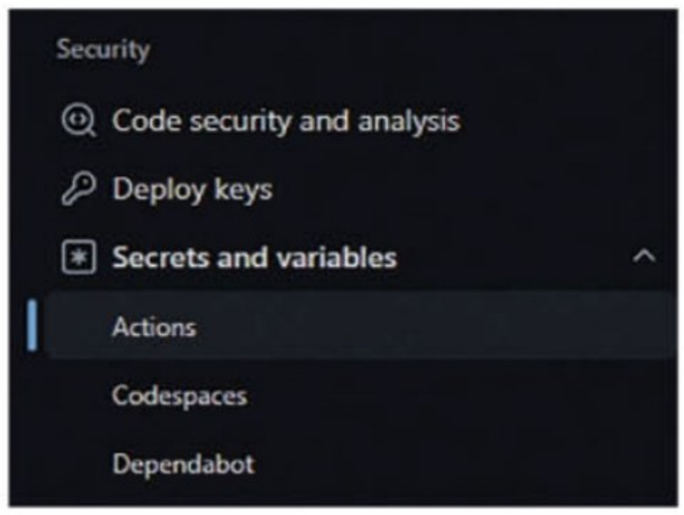


Figure 3

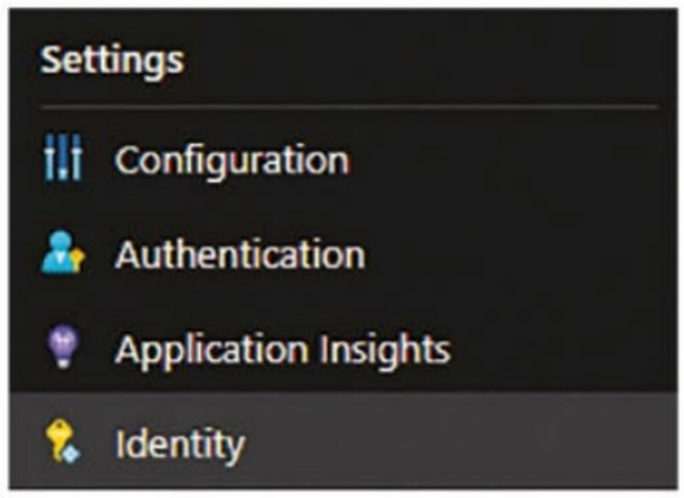


Figure 4

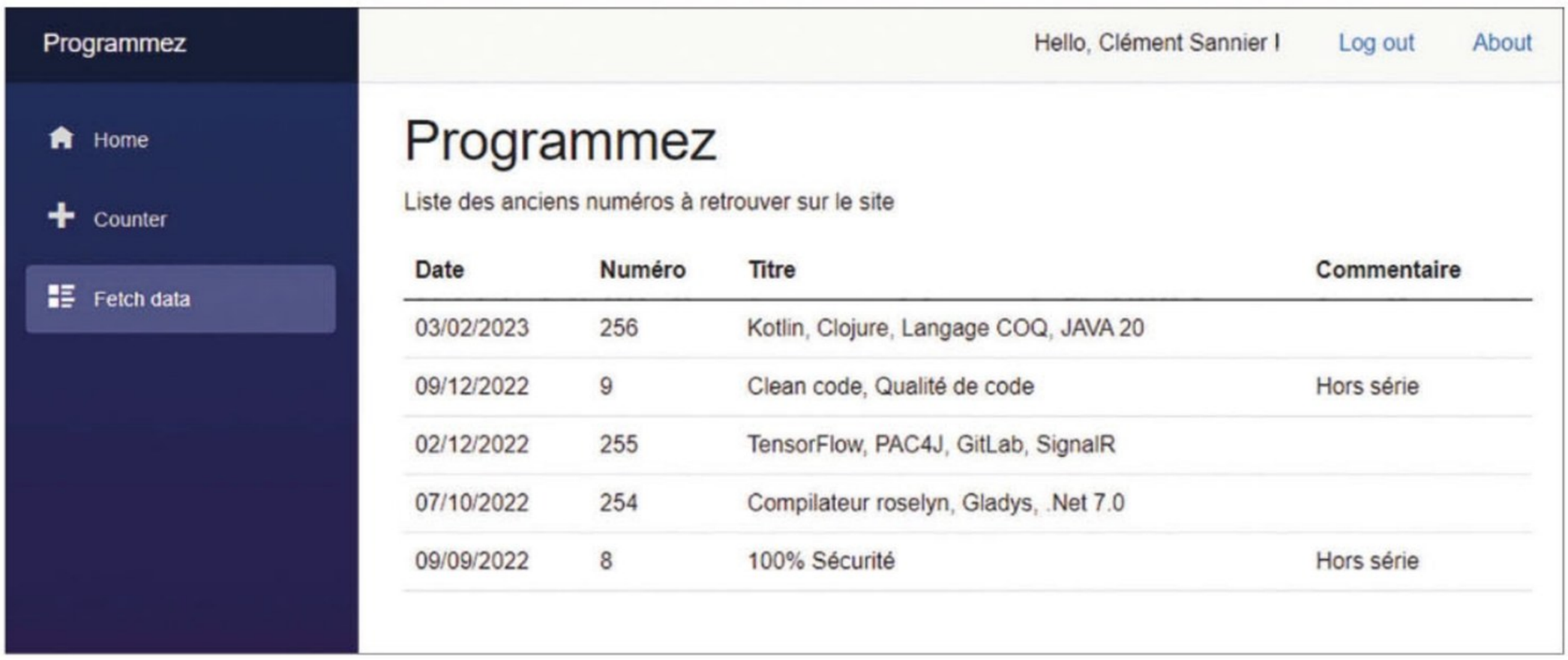
```
az cosmosdb sql role assignment list —account-name cosmos-programmez-csa —
resource-group rg-programmez
```

Il reste une dernière étape à considérer pour accéder à vos données dans votre application Web. Actuellement, nous tentons de nous connecter avec les identifiants de l'émulateur. Il est dès lors nécessaire de modifier la façon dont nous nous connectons à la base de données dans notre code. Voici la modification à apporter :

```
using var cosmosClient = new CosmosClient(account,
        new DefaultAzureCredential());
```

Dans cette nouvelle version, nous n'avons plus besoin d'une clé, mais d'utiliser l'authentification via Azure. Effectuez un git push pour lancer le déploiement automatique et en quelques secondes, vous verrez apparaître la liste des données sur la page "fetchdata".

Pour retrouver mon code, rendez-vous sur : https://github.com/csannierfr/Programmez-AzureSecurity



Cloud : un Idem pour les gouverner tous

Le Cloud est partout (et nulle part), tout le monde y accède et y faire presque n'importe quoi si tant est qu'il dispose du bon compte et des bons accès. Cela rend son exploitation compliquée pour les administrateurs et les équipes sécurité d'entreprise qui se heurtent à ce nouveau défi, surtout qu'il n'y en a pas qu'un type de Cloud et qu'ils n'ont pas d'API ni de services communs (sinon ce ne serait pas marrant).



David Jakubowicz

David est Ingénieur ESIEA, avec plus de 20 ans de métier dans l'avant-vente pour des éditeurs de logiciels. Il est depuis 9 ans Cloud Management Spécialiste chez VMware, spécialisé dans la gestion des Clouds, la supervision et l'automatisation. Il donne aussi des cours sur la Virtualisation et le Cloud Computing (majeure ingénierie du numérique) à l'EPF depuis 5 ans. Il aime coder mais il faut que cela soit rapide et efficace pour servir au mieux les intérêts de ses clients exigeants.

vmware®

Comment imposer (puisqu'il s'agit bien de cela) une norme, une conformité, une sécurité standard, une limitation de dépenses, et/ou une directive d'entreprise en se simplifiant la tâche au maximum ?

Pour cela, on dispose gracieusement d'une interface différente pour chaque cloud et d'un CLI et API… Bref, rien de très simple ni efficace quand il faut lancer des campagnes de vérifications et connaître les API par cœur.

Il y a quelques années, Thomas Hatch, le créateur de la solution open source Salt, dont le but est de faire du config management en décrivant l'état désiré de l'objet, a de nouveau une brillante idée et décida de nous entrainer vers le monde de l'Idempotence.

Idem quoi ?

Idempotent, c'est là la grande idée : on va disposer d'une forme de langage Infrastructure as Data, Idem, qui va nous aider à décrire, créer, modifier, contrôler et activer les objets des différents Clouds. Ce nouvel open source permet de ne pas s'embêter à connaître le code nécessaire pour modifier un objet, car il va le décrire sous format Infrastructure as Data dès qu'on interrogera la cible et donc générer le code pour nous. Il suffira de modifier les valeurs des champs pour obtenir ce que l'on veut comme état désiré et non apprendre un n-ième langage.

Un exemple vaut mille mots :

On va commencer par la base pour bien comprendre le principe d'Idem et les différences avec ce qu'il se fait sur le marché, ici on va modifier une règle de security group AWS, mais cela pourrait être n'importe quel objet d'un Cloud supporté par Idem.

1 On s'assure d'abord que les prérequis sont faits :

- téléchargement de python, de pip, d'idem et des providers pour les destinations (ici AWS)
- création du fichier d'authentification et encryptage de ce dernier.

(voir https://docs.idemproject.io/getting-started/en/latest/topics/gettingstarted/installing.html pour les étapes et détails)

2 On passe alors à la première étape : la notion de description.

```
idem describe aws.ec2.security_group_rule
sgr-012adae8cdb57429f:
  aws.ec2.security_group_rule.present:
  - name: sgr-012adae8cdb57429f
  - resource_id: sgr-012adae8cdb57429f
  - group_id: sg-ba0044db
  - is_egress: false
  - ip_protocol: tcp
  - from_port: 8080
  - to_port: 8080
  - cidr_ipv4: 0.0.0.0/0
  - description: Security group desc
  - tags:
      test_name5: test-rule5
```

Ici on récupère la configuration des objets demandés, à savoir les différentes règles de security_group de ma destination (nota : j'ai filtré le retour pour ne pas prendre trop de place). L'output est effectué à l'écran, donc si on veut récupérer cela dans un fichier, il faudra penser à l'envoyer vers celui-ci (pour usage ultérieur).

3 À cela on va maintenant appliquer des modifications (je change le port TCP accepté en ouverture de 8080 à 8443)

Après avoir sauvegardé le descriptif de la règle de security group, on va juste modifier ses attributs, pas besoin de connaître autre chose que nos datas à changer, le descriptif est déjà fait (le code en fait). Le fichier security_rule.sls contient donc uniquement :

```
sgr-012adae8cdb57429f:
  aws.ec2.security_group_rule.present:
  - name: sgr-012adae8cdb57429f
  - resource_id: sgr-012adae8cdb57429f
  - group_id: sg-ba0044db
  - is_egress: false
  - ip_protocol: tcp
  - from_port: 8443
  - to_port: 8443
  - cidr_ipv4: 0.0.0.0/0
  - description: Security group desc
  - tags:
      test_name5: test-rule5
```

On relance idem pour appliquer les modifications à notre règle de sécurité :

```
idem state security_rule.sls
aws.ec2.security_group_rule_|-sgr-012adae8cdb57429f_|-sgr-012adae
8cdb57429f_|-present:
 ___id___: sgr-012adae8cdb57429f
 ___run_num: 2
 changes:
  new:
   from_port: 8443
   to_port: 8443
  old:
   from_port: 8080
   to_port: 8080
 comment:
 - 'Update from_port: 8443'
 - 'Update to_port: 8443'
 - aws.ec2.security_group_rule sgr-012adae8cdb57429f already exists
 esm_tag: aws.ec2.security_group_rule_|-sgr-012adae8cdb57429f_|-sgr-
012adae8cdb57429f_|-
 name: sgr-012adae8cdb57429f
 new_state:
  from_port: 8443
  to_port: 8443
 old_state:
  from_port: 8080
  to_port: 8080
 rerun_data: null
 result: true
 start_time: '2023-02-07 11:12:27.142487'
 tag: aws.ec2.security_group_rule_|-sgr-012adae8cdb57429f_|-sgr-012
adae8cdb57429f_|-present
 total_seconds: 0.179248
```

Et voilà ! Idem en profite pour nous donner en plus les valeurs avant et après pour suivre les deltas de modifications.

4 On va renforcer la conformité :
Si un utilisateur change dans l'interface AWS ou en API/CLLI les valeurs du port d'entrée du security group (par exemple en changeant mon 8443 et 80), je peux depuis Idem remettre (renforcer) la configuration de départ. Idem va comparer les états et configurations et remettre en place la configuration qui était enregistrée (voir cela comme une sorte de golden model).

Il suffit juste alors de relancer Idem avec le fichier de configuration que nous avons gardé et le tour est joué : c'est le même principe qu'une modification. Si la règle a disparu, elle sera recréée.

```
idem state security_rule.sls
aws.ec2.security_group_rule_|-sgr-012adae8cdb57429f_|-sgr-012adae8cdb57429f_
|-present:
 ___id___: sgr-012adae8cdb57429f
 ___run_num: 2
 changes:
  new:
   from_port: 8443
   to_port: 8443
  old:
   from_port: 80
   to_port: 80
 comment:
 - 'Update from_port: 8443'
 - 'Update to_port: 8443'
 - aws.ec2.security_group_rule sgr-012adae8cdb57429f already exists
 esm_tag: aws.ec2.security_group_rule_|-sgr-012adae8cdb57429f_|-sgr-012adae
8cdb57429f_|-
 name: sgr-012adae8cdb57429f
 new_state:
  from_port: 8443
  to_port: 8443
 old_state:
  from_port: 80
  to_port: 80
 rerun_data: null
 result: true
 start_time: '2023-02-07 16:02:10.409928'
 tag: aws.ec2.security_group_rule_|-sgr-012adae8cdb57429f_|-sgr-012adae8cdb57
429f_|-present
 total_seconds: 0.170944
```

Petite astuce : on peut lancer la commande avec l'option -- test pour voir les changements qui vont s'opérer sans les exécuter.

En conclusion

On dispose maintenant d'Idem comme langage pour exporter les objets cibles sous un format simple, de faire du CRUD et, si une modification est faite dans le dos des administrateurs (via un accès direct API ou CLI), elle peut être remise en état conforme simplement.

Cette solution permet de (re)mettre en place des politiques de sécurité et de conformité depuis un seul point vers tous les Clouds de l'entreprise avec un suivi des dérives et remises en état.

Ajoutez à cela des templates (rangés par catégorie de cas d'usage : le bootstrap, la config, la sécurité, le coût) pour faire les tâches de bases, un ordonnanceur avancé pour tester à intervalles réguliers les états et une interface pour simplifier tout cela, et vous obtenez la solution VMware Aria Guardrails… mais ça, c'est une autre histoire.

Références

Documentation générale sur Idem :
https://www.idemproject.io/,
https://docs.idemproject.io/idem/en/latest/index.html#

Liste des objets et modules disponibles et manipulables par le plugin Idem pour AWS :
https://docs.idemproject.io/idem-aws/en/latest/quickstart/index.html

Une app de gestion EPUB pour les bibliothèques en mode cloud

Le début des années 2010 a marqué un tournant dans le développement web avec les frameworks SPA (Single Page Application) qui ont permis d'offrir la séparation concrète de l'UI et du serveur. À la même époque sont arrivés les premiers clouds : on parle de 2006 pour AWS et 2008 pour Azure. À cette époque les services étaient globalement limités au stockage et à de la machine virtuelle.



Lionel Dimitrijevic

Practice Manager - Neos-SDI

Plus de 20 ans d'expérience autour des technos Microsoft, mes dernières passions sont le cloud, le devops et le développement front.

Neos-SDI an open company

Ces services facilitent le passage au cloud avec des scénarios dits de lift and shift : vous prenez votre infrastructure on premises et vous la portez tel quel ou presque vers des services cloud. C'est en général la première étape vers le cloud. Depuis les plateformes cloud intègrent plusieurs centaines de services PaaS ou SaaS, du simple hébergement de site web à la réalisation d'application low code/no code. La conséquence directe aux développeurs a été un changement dans la conception de solutions applicatives. Débarrassé ou plutôt découplé de l'interface graphique, le serveur traditionnel « monolithique » vit certainement ses dernières années au profit des solutions plus souples que peut offrir le PaaS.

S'il est vrai que l'ingénierie logicielle a évolué au cours de ses 20 dernières années, en passant des architectures N-Tiers vers des architectures plus complexes, mais également plus évolutives telles que les architectures hexagonales, l'arrivée du cloud nous permet aujourd'hui de concevoir des architectures basées sur des services ou des composants. Il ne s'agit pas ici d'architecture microservice, à proprement parlé, mais plutôt de la mise en place d'éléments logiciels réutilisables et qui combinés entre eux vont concrétiser une solution applicative complète. C'est précisément ce que les services PaaS nous permettent de réaliser.

Problématique

Pour illustrer mon propos, nous allons concevoir une petite application de gestion de fichiers EPUB en nous appuyant sur le cloud Azure. Notre application, ou solution logicielle, doit permettre à une bibliothèque de référencer l'ensemble des livres numériques disponibles et d'offrir à ses abonnés une interface leur permettant de visualiser les livres disponibles dans le catalogue. Le format epub est un format qui permet de standardiser les publications numériques et qu'il est aujourd'hui géré par le W3C. Un fichier e-pub est un fichier zip contenant une table des matières, des illustrations, des meta-données sur la publication ainsi bien évidemment que le contenu de celle-ci. Plusieurs fichiers permettent de « décrire » la publication :

- un fichier content.opf au format XML qui décrit la structure de la publication, c'est-à-dire le fichiers constituant les différents chapitres et leur ordonnancement, des méta-données permettant de connaître l'auteur, l'éditeur, d'avoir une description du document
- un fichier toc.ncx décrivant la table des matières de la publication
- un ensemble de fichiers HTML et CSS qui stockent le contenu de la publication et décrivent sa mise en page.
- Des illustrations avec de manière conventionnelle la page de couverture du livre.

Si vous voulez en savoir plus sur le format e-pub, vous pouvez visionner la vidéo de notre émission hebdomadaire du 13 :30 consacrée à ce sujet : https://www.twitch.tv/videos/1666546262 ou aller directement voir les spécifications du format : https://www.w3.org/publishing/epub3/epub-spec.html

Sur la base de ces éléments, notre solution doit offrir une interface d'administration pour que les employés de la bibliothèque puissent référencer les différents ouvrages et une interface de consultation.

Une approche traditionnelle consisterait à mettre en place, par exemple, une app MVC, avec un espace utilisateur et un espace d'administration. L'ensemble des traitements seraient réalisés sur le serveur. On y adjoindrait une base de données pour le référencement des ouvrages et un serveur de fichier pour le stockage des œuvres.

Cette architecture est correcte, mais a plusieurs « défauts » :

- Lorsque mon service va grandir, mon serveur va être de plus en plus sollicité
- Si une bibliothèque indexe un grand nombre de livres, les traitements seront faits sur ce seul serveur et les ressources utilisées pour des tâches administratives pénaliseront les autres utilisateurs
- Notre système est synchrone et ne tient pas compte de potentiels traitements conséquents. Le traitement d'un EPUB peut durer quelques secondes en fonction de ce que l'on souhaite en faire (mise disposition d'un extrait, conversion faire un format autre tel que du PDF, etc.).

Imaginons que cette solution doit pouvoir adresser non pas une bibliothèque, mais un ensemble de bibliothèques, « clientes » de ce service. Il nous faut dans ce cas avoir une capacité de scaling pour répondre aux pics de sollicitation. Nous pourrions scaler cette architecture en multipliant les VM hébergeant le site, cependant, notre mise à l'échelle serait grossière. L'ensemble de notre infrastructure se mettrait systématiquement à l'échelle alors que seules certaines fonctionnalités nécessiteraient une augmentation de capacité. Nous voyons donc que notre architecture n'est pas très résiliente et qu'elle nécessite une adaptation pour répondre à cette logique. Nous pourrions également faire évoluer notre architecture et nous tourner vers la mise en place d'un cluster

de containers, mais nous avons également la possibilité de nous tourner vers des solutions cloud et c'est précisément cette piste que nous allons explorer.

PaaSsons dans le cloud

L'architecture précédente peut être reproduite de manière très simple avec une web App, une base de données Azure SQL et un Storage Blob. **Figure 1**

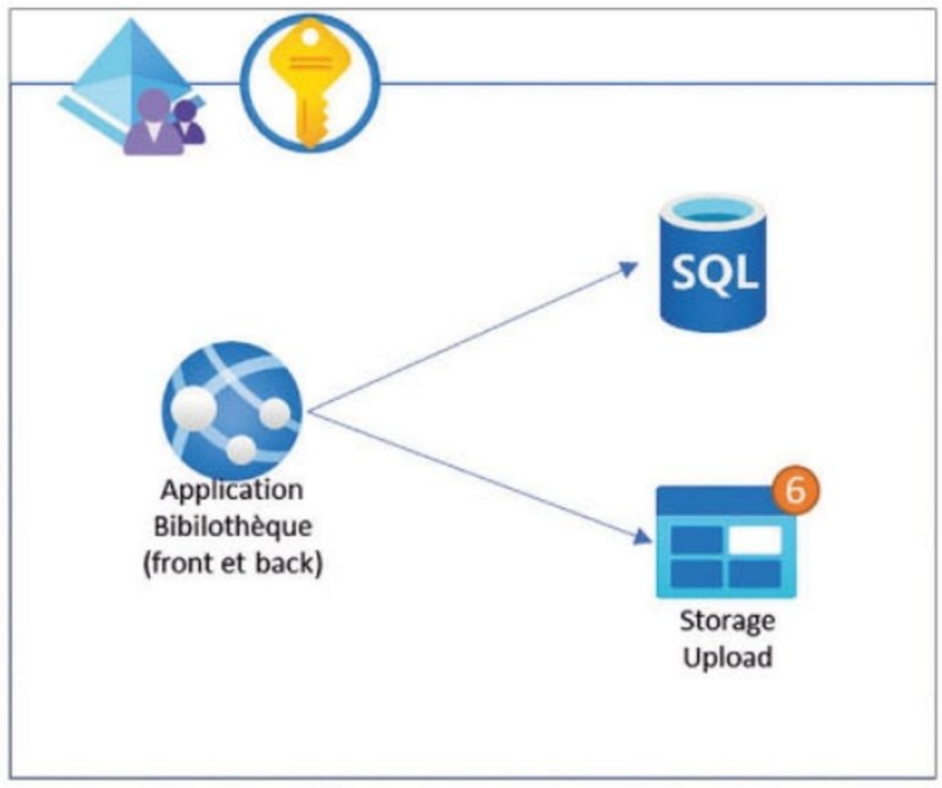


Figure 1

Nous bénéficierions du scaling automatisé, mais les reproches faits au système on-premises demeurent.
Revoyons donc les objectifs à atteindre pour construire avec l'aide des services PaaS Azure une solution qui réponde aux critères suivants :

- Rendre notre solution résiliente
- Adopter un principe de Single Responsability au niveau de notre « infrastructure » : cela permettrait d'avoir des briques applicatives distinctes plus petites et par conséquent avec une maintenabilité accrue
- Bénéficier de services « prêt à l'emploi » et ainsi éviter de dépenser de l'énergie (et du budget) dans la mise en place de serveurs divers et variés. Ce budget non dépensé pourrait être consacré à la réalisation d'un script IaC (Infrastructure as Code) pour mettre en place une véritable démarche DevOps lors de la réalisation de notre solution.
- Bénéficier d'un ensemble de pratiques de sécurité : sécurisation des échanges entre les modules applicatifs, centralisation des configurations applicatives, sécurisation des « secrets » permettant les diverses connexions.

Plusieurs solutions peuvent être apportées, mais nous allons essayer de bâtir, une solution fiable, scalable, extensible et à moindre coût. Par moindre coût il faut entendre un coût au plus proche de ce que nous consommons réellement. Pour cela nous allons utiliser au maximum les services qui proposent une facturation à la consommation :

- Azure Logic Apps : ce service va nous servir d'orchestrateur. Il offre l'avantage de ne nécessiter presque aucune ressource en programmation puisqu'il s'agit de modéliser un workflow avec des briques prêtes à l'emploi. Avantage majeur ici, la rapidité de mise en œuvre ! En moins d'une demi-journée de travail, il est possible de mettre en place tout le mécanisme d'orchestration, de faire exécuter certaines tâches en parallèle pour optimiser le temps de traitement.
- Azure Functions : ce service serverless va nous permettre d'exécuter du code sans nous soucier d'une quelconque plateforme sous-jacente. Par ailleurs, le scaling est automatique ce qui nous garantit de pouvoir répondre à toutes les demandes sans nous soucier de quoi que ce soit. Nous pouvons ainsi concentrer 100% de nos efforts sur le code. L'exécution de ces fonctions peut se faire grâce à des déclencheurs variés (temps, appel HTTP, modification d'un storage, etc.). Attention cependant, à ne pas tout demander aux Azure Function. Le niveau de facturation étant calculé au temps d'exécution, il convient d'optimiser le code exécuter et finalement de faire des tâches relativement unitaires. Cela convient parfaitement dans le processus que nous avons défini pour notre intégration de fichiers EPUB.

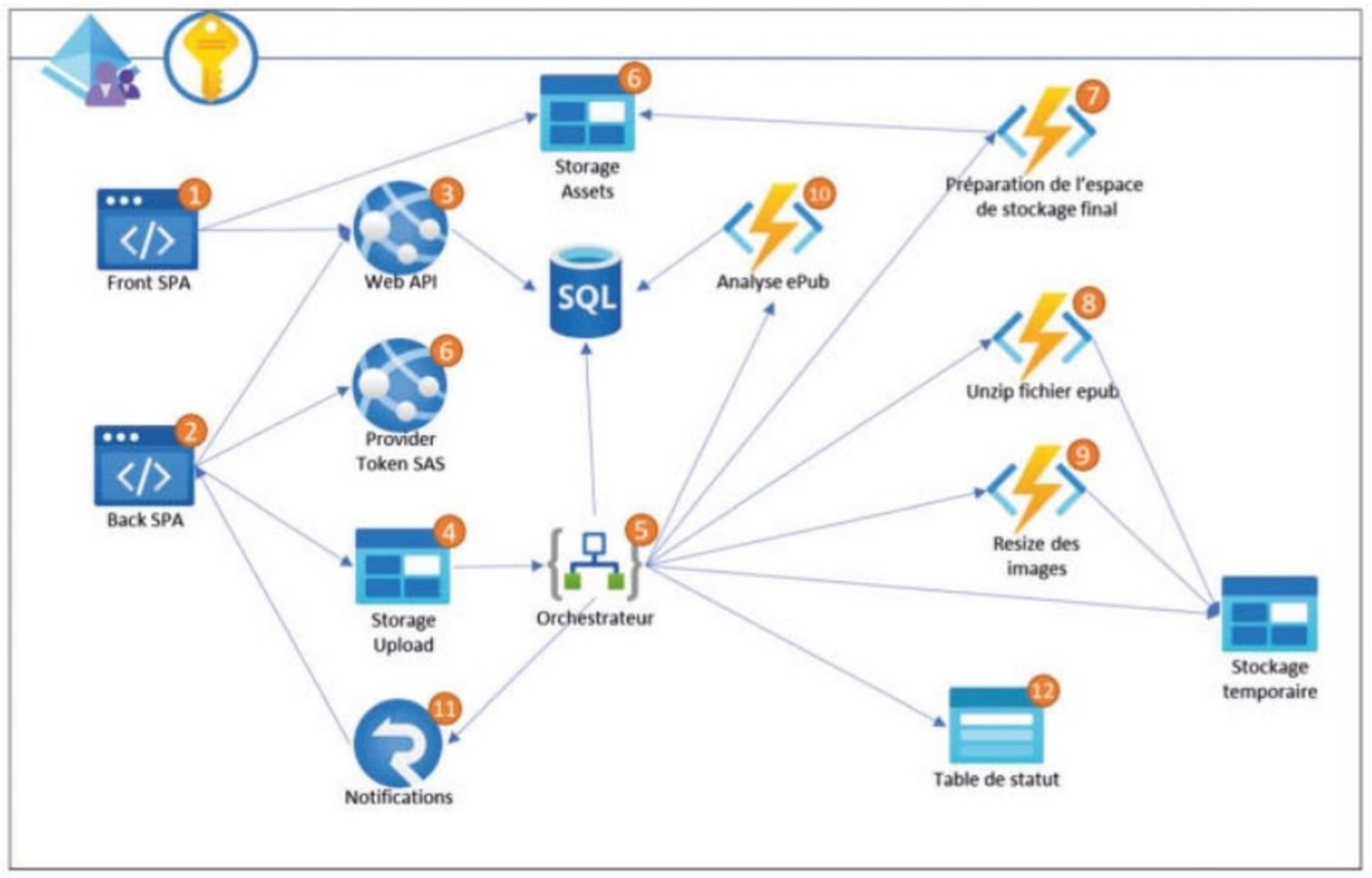


Figure 2

- Nous utiliserons également quelques Storage pour pouvoir stocker nos fichiers EPUB, mais aussi pour avoir des espaces de dépôt temporaire pour dézipper les fichiers ou stocker les images de couverture de livre retaillées aux bonnes dimensions. Le coût du stockage étant minime et in fine proportionnel à ce qui y est déposé cela répond parfaitement à l'objectif de maîtrise des coûts énoncés.
- D'autres services indispensables devront aussi être mis en œuvre pour assurer un minimum de sécurité, pour héberger notre WebAPI et pour stocker nos données. En ce qui concerne le stockage de données, nous pourrons nous tourner vers une base de données facturée à la consommation. Attention cependant, d'expérience il est possible de très vite dépasser les budgets prévus. Il faudra alors plutôt se tourner vers une facturation au DTU ce qui correspond peu ou prou à avoir un certain nombre de ressources disponibles en échange d'une facturation fixe. En cas de faible utilisation, ce modèle peut en revanche s'avérer moins intéressant.

À partir de ces éléments, nous pourrions envisager l'architecture suivante : **Figure 2**

L'usage du conditionnel est important, car il n'y a souvent pas une seule vérité lorsque l'on conçoit une architecture dans le cloud. Nous aurions tout aussi bien pu utiliser des API classiques hébergées sur une WebApp plutôt que des Azure

Functions. Nous aurions également pu nous passer de la Logic Apps en utilisant des « fonctions durables », etc.
Concevoir dans le cloud c'est en permanence faire des choix entre coûts (en €), praticité de mis en œuvre, mais aussi coût de réalisation.

Détail de la solution

Nos deux apps SPA (front et back) sont hébergées sur des espaces de stockage Blob (1 et 2) avec l'option « Static web site » activée. Cela permet de pouvoir distribuer notre application au prix du stockage puisque pour une SPA nous n'avons pas besoin de serveur applicatif. Autre avantage, nous ne surchargeons pas notre serveur d'API (3), qui est lui hébergé sur un service de type web App. Notre application back va avoir besoin d'uploader les fichiers ePub sur un autre storage blob (4). C'est l'évènement de dépôt d'un nouveau fichier sur ce storage qui va déclencher notre orchestrateur Logic Apps (5). Pour pouvoir accéder depuis notre SPA au storage d'upload nous allons avoir besoin d'avoir un token SAS (shared access signature). En effet, pour accéder au storage, nous pouvons soit l'ouvrir au monde et ne pas mettre en place de sécurité particulière, soit le sécuriser pour éviter des écritures indues. Dans le premier cas, tout le monde pourra accéder aux données du storage ce qui est utile dans un scenario où l'on fourni des éléments statiques en lecture seule à un site web, comme pour notre storage (7) permettant à la SPA front d'accéder aux fichiers EPUB et aux images de couverture des ouvrages. En revanche, dans le cas d'une écriture sur un storage, il convient d'en sécuriser l'accès.
Cette sécurisation peut se faire via un partage de clé ce qui n'est pas envisageable avec une application SPA, car notre clé pourrait être récupérée par un utilisateur mal intentionné. L'autre solution consiste à générer des token d'accès avec une durée d'utilisation très courte pour nous permettre d'effectuer une simple opération d'upload. C'est exactement le rôle de notre web App (6). Le provider de token pourra être hébergé sur le même plan que notre web API (3) ou bien avoir son propre plan. Un plan peut être conceptualisé comme un serveur. Le fait d'héberger nos deux web App (3 et 6) sur le même plan nous permet de minimiser les coûts, car c'est le plan qui est facturé et non le service web App. En revanche, nous perdons légèrement en souplesse : il est en effet peu probable que notre service de token soit aussi sollicité que notre API et il pourrait être envisagé de donner plus de puissance à l'API plutôt qu'au fournisseur de token. En fonction du scénario, la mise en place d'un plan dédié pour notre fournisseur de token pourra être envisagé. Comme toujours lors de la mise en place d'une solution cloud, il faut contre-balancer le coût en fonction du service rendu et de la facilité de mise en œuvre.
Avant d'uploader un fichier EPUB dans la solution, l'application back va demander un token au service ad hoc et utiliser ce dernier pour pouvoir déposer le fichier sur le storage correspondant.
Une fois ce dépôt effectué, la Logic Apps (5) prend le relais et va fonctionner tel un orchestrateur pour exécuter les différentes tâches nécessaires en lançant les Azure Functions correspondantes (7, 8, 9 et 10). Chacune d'elle va exécuter une tâche dédiée et offrira l'avantage de pouvoir être éventuellement réutilisée dans un autre scenario, ce qui peut être le cas pour notre fonction Dezip. Pour suivre la réalisation des différentes étapes et leur statut, un storage de type table est mis en œuvre (12). Nous aurions pu aussi nous appuyer sur là-bas de données SQL, mais inutile de mélanger des données de traitement et des données « fonctionnelles » dans la même base de données. Qui plus est, le coût de notre Table Storage sera minime puisque ce service fait partie du service Storage (au même titre que notre stockage blob). Notre client back pourra se connecter à cette table de statut pour voir l'avancée du traitement en temps réel au besoin, sans surcharger notre base de données principale sur laquelle beaucoup d'utilisateurs pourront être amenés à se connecter. Nous séparons ainsi les problématiques de la même manière que nous le faisons lorsque nous concevons un logiciel.
Le processus de traitement des fichiers EPUB est réalisé de manière complètement asynchrone. Cela présente l'avantage de ne pas bloquer l'utilisateur plus longtemps que le temps nécessaire à l'upload du fichier. Son expérience utilisateur n'en sera que meilleure. Cependant il est de bon ton de prévenir l'utilisateur de la fin du traitement. En effet, même si la table de suivi des intégrations peut être interrogée et suivie à l'aide d'une interface graphique dans l'application back, être notifié directement est quand même bien plus ergonomique. Une fois les différentes tâches réalisées la Logic Apps va par conséquent notifier l'application back du bon référencement de notre fichier EPUB dans le système. Pour cela, elle va solliciter une nouvelle fonction Azure dont le rôle va simplement être de publier un message vers le service SignalR (11) afin que ce dernier relaie l'information au client.

Un peu de sécurité

Nous allons devoir nous appuyer sur plusieurs briques Azure. Tout d'abord en ce qui concerne les parties exposées (WebAPI et applications clientes) ainsi que pour l'authentification et la gestion des utilisateurs, nous nous appuyons sur Azure AD B2C. Ce service nous permet de déclarer et sécuriser nous applicatifs avec un flux OAuth tel que décrit dans le schéma suivant : **Figure 3**

Azure B2C nous permet de gérer l'intégralité du workflow d'authentification très simplement. Il suffit de référencer les applications à protéger, de mettre en place du côté des applications les bibliothèques adéquates, d'appliquer la configuration et l'ensemble se met à fonctionner immédiatement. Par défaut, les pages d'authentification sont celles de Microsoft, mais il est possible de les personnaliser et de les « brander » assez facilement avec de l'HTML et un peu de CSS. Azure B2C propose également la prise en charge de l'authentification via des providers tiers tels que Facebook, Google, Twitter, GitHub, etc. Là encore la procédure de prise en charge est très simple et nécessite uniquement de la configuration. Nous avons donc la possibilité pour l'application Front de laisser les utilisateurs s'authentifier soit en utilisant un login/mot de passe de manière traditionnelle soit d'utiliser un des comptes des providers disponibles ce qui est très courant aujourd'hui.
Pour la sécurisation des briques applicatives « internes » et leur communication nous nous appuyons sur un mécanisme

d'Azure permettant aux services de s'identifier : les managed identities. En effet, chaque service créé dans Azure possède sa propre identité. Nous sécuriserons donc les divers accès entre services avec les identités managés. Une identité managée est une identité qui est directement gérée par Azure. Nous n'avons par à stocker de login ou de mot de passe, on ne les connaît pas, et nous n'avons pas besoin de les connaître. Ce sont ces identités qui vont permettre aux applications d'accéder à des données du key vault par exemple. Deux types d'identités managées sont disponibles dans Azure :

- Système assigned : Il s'agit d'une identité propre à la ressource Azure qui possède le même cycle de vie. Quand la ressource ou le service est supprimé, l'identité l'est aussi.
- User assigned : Il s'agit ici d'une identité distincte de toute ressource Azure. Elle peut être partagée entre plusieurs services et peut être vue comme des identités d'application et non de ressource comme les identités « system assigned ». Il faut noter que le cycle de vie des identités managées « user assigned » est indépendant de toute ressource et qu'il faudra la supprimer manuellement le cas échéant.

Toutes ces briques de sécurité sont fournies nativement par la plateforme Azure et ne nécessitent aucune implémentation particulière. Un peu de configuration et de compréhension de système permettent à chacun de sécuriser sa solution. Certes cette sécurisation est sommaire, mais comparée à ce qui est fait traditionnellement par des développeurs sur un serveur IIS par exemple, c'est sans commune mesure.

Un autre élément de sécurité est mis en place : Azure Key Vault. Ce service va nous servir à stocker tous les éléments critiques de notre solution : chaînes de connexions, certificats, etc. Cette ressource est indispensable pour tous les projets réalisés dans le cloud et doit être considérée comme une bonne pratique.

Conclusion

Comme nous avons pu le voir, mettre en place une solution dans le cloud s'avère similaire à la conception d'une application. La mise en place des fonctions Azure s'avère très proche de la mise en place d'une classe et facilite la séparation des problématiques. D'autres enjeux interviennent néanmoins: le coût du service, mais également le coût de mise en œuvre. L'utilisation des services no-code permet de réduire le temps d'implémentation, qui aurait été plus long si nous avions dû tout écrire manuellement. En ce sens, les solutions low code/no code ne sont pas des « ennemis » du développeur, mais bien au contraire une palette d'outils supplémentaire. Plus généralement développer dans le cloud permet au développeur d'avoir une approche similaire à ce qu'il peut faire « naturellement », mais cela apporte en plus des outils de structuration qu'il n'a pas nécessairement lors de la réalisation d'une solution dans un environnement traditionnel. Cela implique d'approfondir son approche : là où existent peut-être une ou deux solutions avec une approche classique, le cloud offre une multitude d'options. Nous avons pris ici le parti de coût, mais nous aurions pu également prendre un autre point de vue qui aurait pu être d'apporter une disponibilité importante de nos services à travers le monde. Les ressources utilisées auraient certainement été différentes avec pourquoi pas du Cosmos DB comme base de données, la mise en place d'un CDN, etc. La seule limite in fine est l'imagination du concepteur/développeur pour répondre aux contraintes qui lui sont posées.

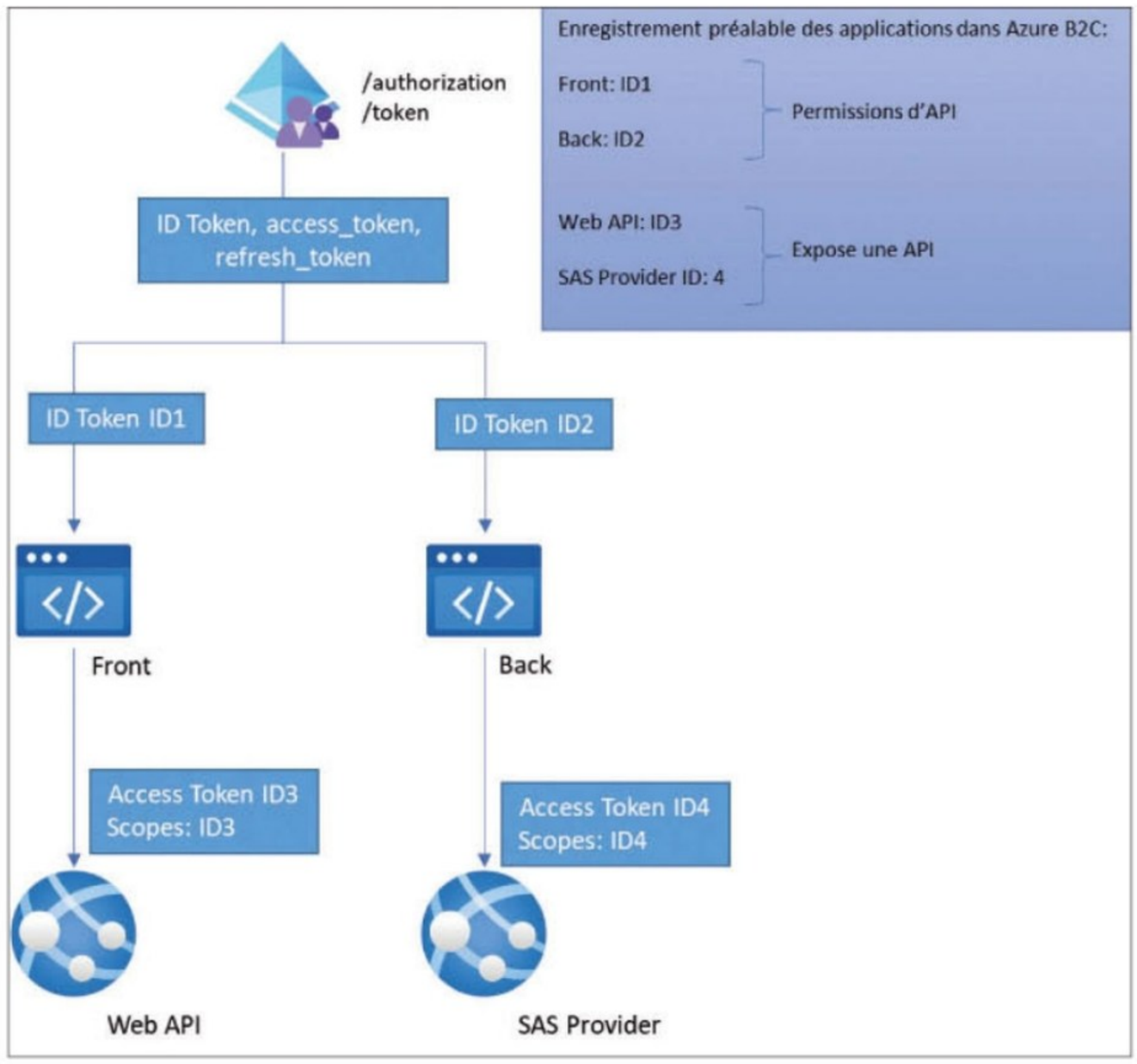


Figure 3

ABONNEMENT NUMÉRIQUE

1 an ..... 45 €

Abonnez-vous sur :
www.programmez.com

FAITES VOTRE VEILLE TECHNOLOGIQUE AVEC

PROGRAMMEZ!

ABONNEMENT PAPIER

1 an ..... 55 €
2 ans ..... 90 €

Voir page 5

Intégration et livraison continues sécurisées sur GCP

PARTIE 1 : GOOGLE CLOUD BUILD, SKAFFOLD ET GOOGLE CLOUD DEPLOY



Laurent Grangeau

Solutions Architect chez Google.

Il est également l'organisateur du meetup Serverless Paris, ainsi que co-organisateur des meetups Kubernetes et Docker Paris. Il a développé dans plusieurs langages, principalement orientés objet comme Java ou C#. Depuis plusieurs années, il aide les entreprises à adopter le cloud, ainsi que les principes DevOps. Il a de l'expérience dans la construction de microservices et de systèmes distribués. Il aime automatiser les choses et exécuter des applications distribuées à l'échelle.

Dans ce dossier en plusieurs parties, nous allons découvrir comment mettre en place de l'intégration continue et de la livraison continue avec GCP. Nous allons pour cela utiliser Cloud Build et Cloud Deploy afin de packager et déployer une application sur Cloud Run.

Je tiens à remercier Olivier Dupré, Tony Jarriault ainsi que Sandra Parlant pour la relecture de cet article, ainsi que leurs précieux conseils.

De nos jours, les attaques sur les Software Supply Chain sont de plus en plus nombreuses. Elles sont une menace croissante pour la sécurité informatique. Ces attaques visent à compromettre le logiciel à un stade précoce du développement, du stockage ou de la distribution, plutôt que de cibler directement les utilisateurs finaux. Elles peuvent entraîner une vulnérabilité à long terme pour les utilisateurs, car le logiciel contaminé peut être diffusé sur de nombreux systèmes avant d'être détecté. **Figure 1**

Nous allons donc mettre en place des vérifications de sécurité lors de la création du package avec Artifact Registry, mais aussi lors du déploiement avec Binary Authorization.

Mais avant d'aborder le vif du sujet, revenons sur la définition de l'intégration continue et la livraison continue.

Intégration et livraison continues (CI/CD)

L'intégration continue (CI) est un processus de développement de logiciels qui consiste à automatiser la construction, le test et le packaging de code. Le but est d'assurer la qualité du logiciel et de détecter les erreurs et les problèmes le plus tôt possible dans le processus de développement. En utilisant une chaîne de CI, les équipes de développement peuvent intégrer fréquemment leur code, c'est-à-dire mettre en commun des parties de code développées indépendamment, et exécuter des tests automatisés pour vérifier la qualité et la cohérence du logiciel. Les résultats de ces tests sont communiqués aux développeurs afin qu'ils puissent corriger les erreurs rapidement.

La livraison continue (CD), quant à elle, est un processus de déploiement de logiciels qui permet de déployer rapidement et fréquemment des mises à jour de code en production. Elle s'appuie sur l'intégration continue pour automatiser les tests et les vérifications de qualité avant le déploiement. Elle implique que les déploiements se fassent "as-code", c'est-à-dire qu'ils soient déclarés dans du code pour pouvoir être automatisés. La livraison continue vise à minimiser le temps nécessaire pour déployer de nouvelles fonctionnalités et corriger les bugs, en automatisant les étapes du processus de déploiement. Cela permet aux équipes de développement de déployer fréquemment de petites mises à jour de code plutôt que de gros paquets de fonctionnalités tous les mois ou tous les ans.

Google Cloud Platform (GCP) offre plusieurs services pour le CI / CD. Par exemple, Google Cloud Build est un service d'intégration continue qui permet de créer et de déployer des applications sur GCP en utilisant des fichiers de configuration de construction. Google Cloud Deploy est un service de déploiement automatisé qui permet de déployer des ressources sur GCP en utilisant des fichiers de configuration de déploiement.

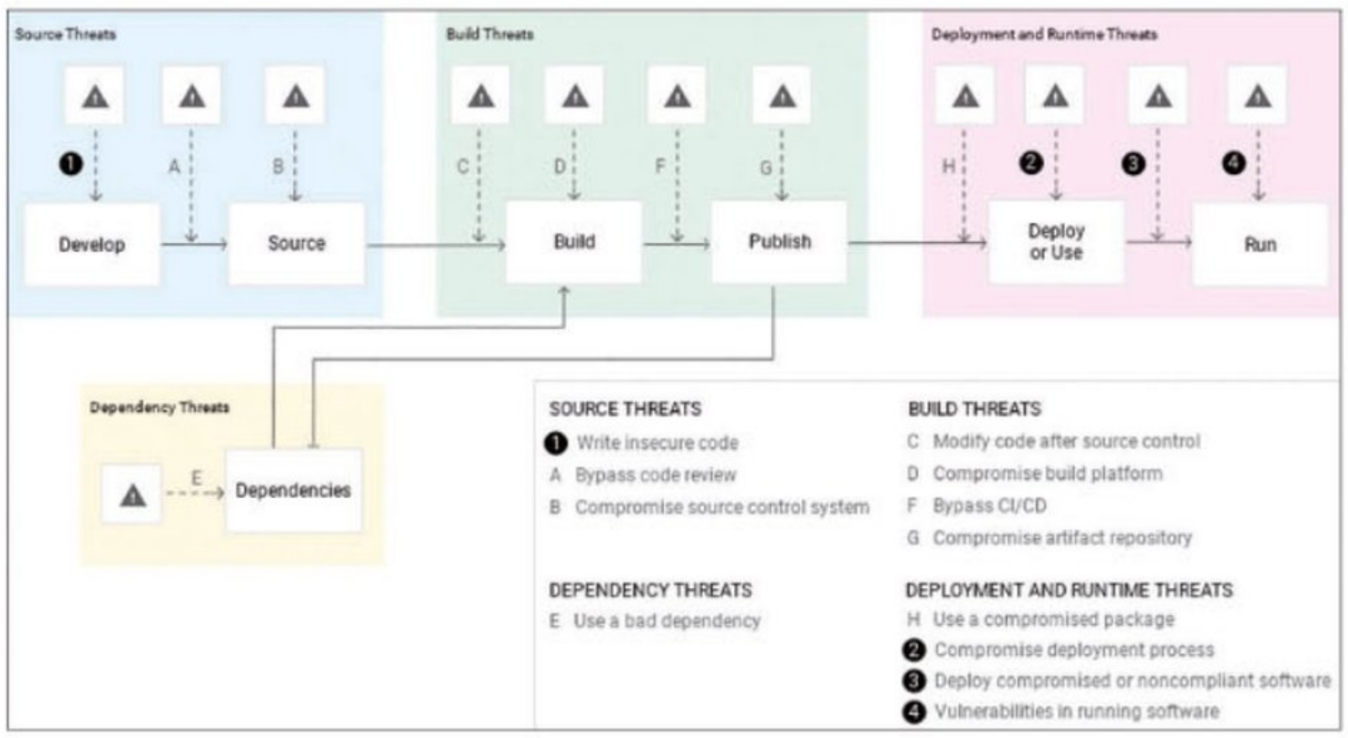
Dans la suite de cet article, nous allons mettre en place de l'intégration continue sur une application exemple, beaucoup utilisée pour les démonstrations Java, qui s'appelle PetClinic. PetClinic est un projet open source de démonstration d'application web Java. Il a été développé en utilisant les frameworks Spring et Hibernate, et il montre comment utiliser ces technologies pour implémenter une application de gestion de clinique vétérinaire. La connaissance de Java n'est pas un pré-requis nécessaire pour la suite de cet article.

Son code source est accessible sur : https://github.com/spring-projects/spring-petclinic

Pour les besoins de l'article, nous allons créer un fork disponible à cette adresse afin de mettre en place les fichiers nécessaires : https://github.com/laurentgrangeau/spring-petclinic

Nous allons aussi utiliser beaucoup de services différents de GCP. Je considère que le lecteur a accès à un projet GCP où faire toutes ces manipulations. Sinon, il existe un tier gratuit afin de pouvoir tester les manipulations sur un projet créé pour l'occasion. Dans mes exemples, j'utiliserai mon projet **lgu-demos**.

Figure 1



Google Cloud Build

La première étape consiste à mettre en place de l'intégration continue avec Cloud Build.

Cloud Build permet aux développeurs de créer et de déployer des applications sur GCP en utilisant des fichiers de configuration de construction. Il est conçu pour faciliter le processus de construction, de test et de déploiement des applications, tout en garantissant la qualité du code.

Avec Google Cloud Build, les développeurs peuvent automatiser les tâches de CI, telles que la compilation de code, l'exécution de tests unitaires et le packaging d'applications, en utilisant des outils populaires tels que Maven, Gradle ou Docker. Il permet également d'intégrer des outils de tests tels que JUnit ou Sélénium pour s'assurer que les modifications apportées au code ne causent pas de régressions.

Une des principales caractéristiques de Google Cloud Build est sa flexibilité. Il permet aux développeurs de choisir les outils de construction qu'ils préfèrent utiliser, et de créer des workflows de construction personnalisés en utilisant des fichiers de configuration de construction. Les fichiers de configuration de construction sont écrits en YAML et décrivent les étapes de construction, les images utilisées, les variables d'environnement et les dépendances.

Google Cloud Build est intégré à d'autres services de GCP, tels que Source Repositories, Stackdriver et Cloud Logging, pour fournir une vue complète du processus de construction et de déploiement. Il permet aux équipes de suivre les performances des builds, de recevoir des alertes en cas d'échec et de visualiser les journaux associés pour résoudre les problèmes éventuels.

Le repository PetClinic ne contenant pas de Dockerfile, nous allons utiliser les Buildpacks afin de détecter automatiquement le langage de développement utilisé et ainsi compiler et packager notre application.

Les Buildpacks permettent de convertir un code source en une image de conteneur sécurisée, efficace et prête pour la production, sans Dockerfile. Ils fournissent un cadre et un support d'exécution pour les applications. Ils examinent les applications pour déterminer toutes les dépendances dont elles ont besoin et les configurent de manière appropriée pour qu'elles s'exécutent sur n'importe quel cloud. **Figure 2**

Il faut aussi activer Artifact Registry afin de pouvoir stocker notre image packagée. Pour ce faire, il faut d'abord activer le service sur GCP avec la commande :

```
gcloud services enable artifactregistry.googleapis.com
```

Ensuite, il faut créer une registry qui contiendra toutes les versions de notre application packagée, au format conteneur.

```
gcloud artifacts repositories create spring-projects —repository-format=docker —location=europe-west1
```

Une fois les commandes effectuées, nous devrions avoir une registry accessible sur **europe-west1-docker.pkg.dev/[PROJECT]/spring-projects/**.

Nous n'allons pas nous attarder sur Artifact Registry maintenant, car nous activerons les scans de vulnérabilités et détaillerons ses fonctionnalités plus tard dans l'article.

Une fois la registry créée, nous pouvons lancer notre premier build Cloud build. Dans un premier temps, nous allons le faire via la ligne de commande qui va s'occuper de tout. Il faudra aussi lui préciser dans la ligne de commande la version de JAVA utilisé avec la variable d'environnement **GOOGLE_RUNTIME_VERSION**.



Figure 2

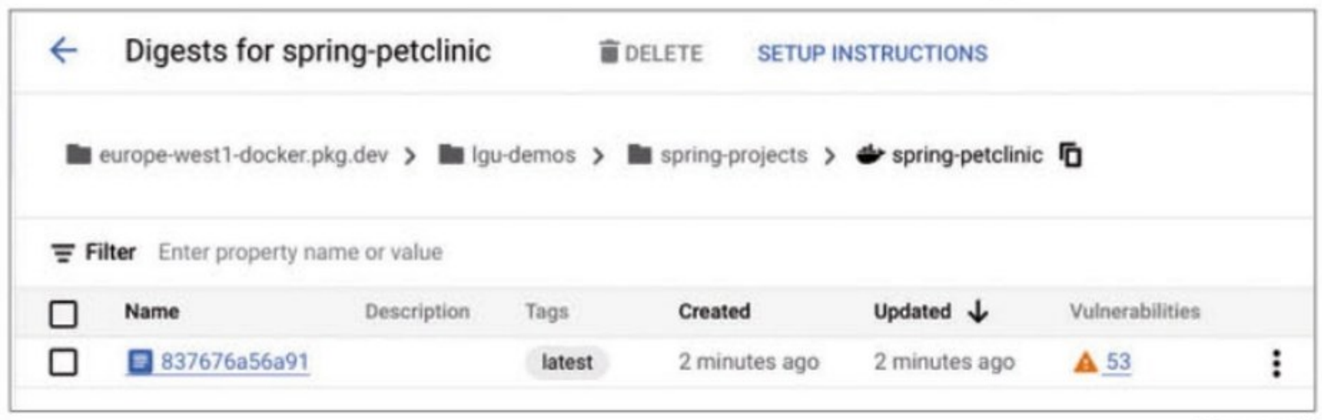


Figure 3

```
gcloud builds submit —pack image=europe-west1-docker.pkg.dev/lgu-demos/spring-projects/spring-petclinic,env=GOOGLE_RUNTIME_VERSION=17
```

Une fois la commande terminée, nous devrions avoir notre premier container dans la registry. **Figure 3**

Cependant, le but de l'intégration continue est de pouvoir compiler, tester et packager l'application à chaque fois que du code source est poussé sur notre repository. Il serait donc plus intéressant de créer un fichier qui contiendrait notre pipeline d'intégration continue. Par convention, Cloud build analyse le contenu du fichier **cloudbuild.yaml** servant à décrire les étapes du pipeline. Nous allons donc créer notre premier fichier **cloudbuild.yaml** à la racine du repository et y ajouter les commandes afin de builder et packager notre application avec Buildpacks.

```
steps:
- id: Build the application
  name: 'buildpacksio/pack'
  args: ['build', '—builder=gcr.io/buildpacks/builder', '—env', 'GOOGLE_RUNTIME_VERSION=17', 'europe-west1-docker.pkg.dev/lgu-demos/spring-projects/spring-petclinic']
- id: Push the container
  name: 'gcr.io/cloud-builders/docker'
  args: ['push', 'europe-west1-docker.pkg.dev/lgu-demos/spring-projects/spring-petclinic']
```

Dans ce fichier, deux commandes sont lancées. La première est l'exécution du Buildpacks qui va détecter le langage de notre application et en créer une image. La seconde est le push de l'image générée dans la registry GCP. Une fois le fichier créé, nous allons configurer un trigger qui va se déclencher à chaque fois que du code sera poussé sur la branche **main**. Pour connecter son repository, il est possible de suivre la documentation disponible ici : https://cloud.google.com/build/docs/automating-builds/github/connect-repo-github

```
gcloud builds triggers create github —name="petclinic-trigger-main" —branch-pattern="^main$" —build-config="cloudbuild.yaml" —repo-owner="laurentgrangeau" —repo-name="spring-petclinic" —region="europe-west1"
```

Maintenant que le trigger a été créé, nous pouvons pousser notre fichier **cloudbuild.yaml** sur le repository GitHub. Le

build de l'application et son packaging vont alors se lancer. Une fois l'opération terminée, Cloud build poussera alors l'application packagée dans la registry configurée. **Figure 4**

Skaffold

Nous avons maintenant un pipeline d'intégration continue avec Cloud build qui nous permet de construire une image applicative et la pousser sur une registry automatiquement à chaque fois que du code source est poussé sur la branche main. Avant de déployer, nous allons légèrement modifier notre pipeline afin d'utiliser Skaffold.

Skaffold est un outil open source développé par Google pour faciliter le processus de développement, de test et de déploiement des applications conteneurisées. Il est conçu pour automatiser les tâches répétitives liées à la construction, au déploiement et au débogage des applications conteneurisées, tout en garantissant la qualité du code.

Avec Skaffold, les développeurs peuvent automatiser les tâches de construction, telles que la compilation de code, l'exécution de tests unitaires et le packaging d'applications, en utilisant des outils populaires tels que Docker. Il permet également d'intégrer des outils de tests tels que JUnit et Sélénium pour s'assurer que les modifications apportées au code ne causent pas de régressions. Skaffold prend en charge plusieurs types de déploiements, notamment les déploiements sur des clusters Kubernetes, ou sur des plateformes cloud native comme GCP. Il peut également déployer automatiquement des applications sur des plateformes telles que App Engine, Kubernetes Engine et Cloud Run. Il permet notamment une grande flexibilité en permettant de déployer sur plusieurs plateformes cloud ou différents types de clusters.

Skaffold est intégré à d'autres outils populaires tels que Helm, Kustomize et Draft, pour fournir une vue complète du processus de construction et de déploiement. Il permet aux équipes de suivre les performances des builds, de recevoir des alertes en cas d'échec et de visualiser les journaux concernés pour résoudre les problèmes éventuels.

Dans un premier temps, nous allons créer un fichier **skaffold.yaml** qui va indiquer à Cloud build comment packager l'application. Skaffold supporte évidemment les buildpacks. Nous allons donc mettre dans ce fichier **skaffold.yaml** les lignes suivantes :

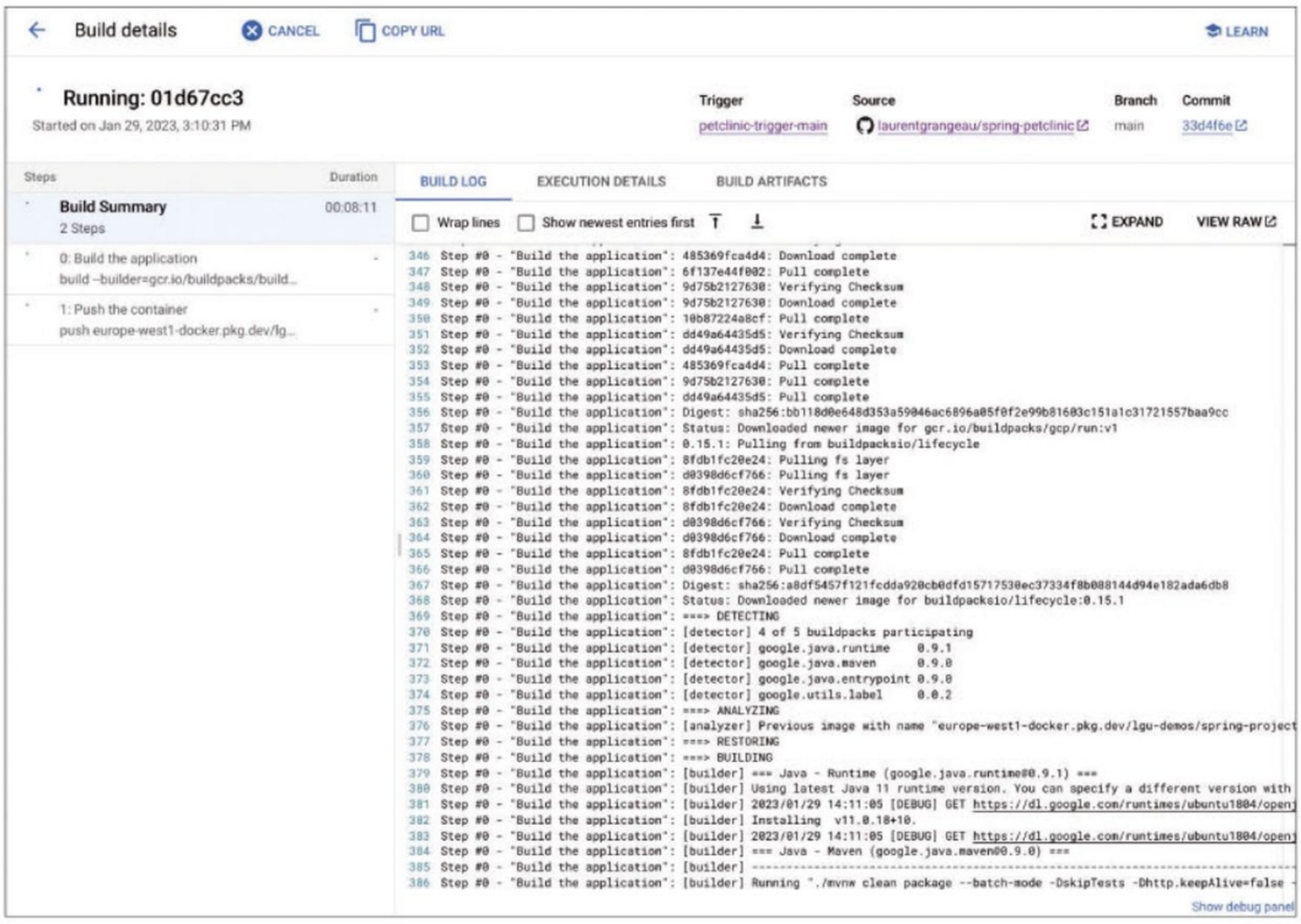
```
apiVersion: skaffold/v4beta2
kind: Config
build:
 artifacts:
  - image: spring-petclinic
    buildpacks:
     builder: "gcr.io/buildpacks/builder:v1"
     env:
      - GOOGLE_RUNTIME_VERSION=17
```

Ces lignes reprennent le principe de notre premier **cloudbuild.yaml**, c'est-à-dire l'utilisation de Buildpacks. Le gros avantage de Skaffold est qu'il peut builder et pousser notre image générée en une seule étape. Il va falloir aussi modifier légèrement notre fichier **cloudbuild.yaml** afin de prendre en compte ce nouveau fichier.

```
steps:
- id: Build the application
  name: 'gcr.io/k8s-skaffold/skaffold:slim'
  args: ['skaffold', 'build', '—default-repo', 'europe-west1-docker.pkg.dev/lgu-demos
/spring-projects']
```

Une fois ces modifications faites, Skaffold va alors builder et pousser votre application packagée dans la registry automatiquement.

Figure 4



Google Cloud Deploy

Nous avons donc maintenant un nouveau fichier **skaffold.yaml** qui pour l'instant ne sert qu'à builder et pousser notre application dans la registry. Cependant, Cloud Deploy, qui est le service managé de livraison continue de GCP, se sert de ce fichier afin de savoir quoi déployer et comment. Google Cloud Deploy est le service de livraison continue de GCP. Il permet de déployer des applications en supportant plusieurs méthodes comme les Buildpacks, les images conteneurisées, mais aussi le code source de l'application. Cloud Deploy supporte une variété de cibles de déploiements comme App Engine, des clusters Kubernetes, mais aussi Cloud Run. Il est basé sur Cloud Build et utilise donc le même service sous-jacent. Cloud Deploy est un service géré qui automatise la livraison de vos applications à une série d'environnements cibles dans une séquence de promotion définie. Lorsque vous souhaitez déployer votre application mise à jour, vous créez une version, dont le cycle de vie est géré par un pipeline de livraison. Dans un premier temps, il faut définir un pipeline de déploiement. Il servira à Cloud Deploy afin de savoir dans quel ordre promouvoir l'application. Ce pipeline se base sur des Targets, qui sont des environnements cibles où déployer notre application. Vous trouverez ci-dessous un exemple de pipeline de déploiement qui va pousser notre application sur un environnement de développement Cloud Run. Ce fichier doit être nommé **clouddeploy.yaml**.

```
apiVersion: deploy.cloud.google.com/v1
kind: DeliveryPipeline
metadata:
 name: cloud-run-pipeline
description: Spring PetClinic deployment pipeline
serialPipeline:
 stages:
 - targetId: dev-env
  profiles: [dev]
---
apiVersion: deploy.cloud.google.com/v1
kind: Target
metadata:
 name: dev-env
description: Cloud Run development service
run:
 location: projects/lgu-demos/locations/europe-west1
```

Une fois ce fichier créé, il faut indiquer à Cloud Deploy comment créer le pipeline. Pour ce faire, il suffit d'utiliser la commande ci-dessous afin de créer le pipeline de déploiement sur GCP :

```
gcloud deploy apply —file=clouddeploy.yaml —region=europe-west1 —project
=lgu-demos
```

Nous nous retrouvons alors avec un premier pipeline de livraison continue dans Cloud Deploy. **Figure 5**

Pour pouvoir déclencher ce pipeline à la suite de la création de notre image applicative, il faut alors ajouter à la suite de notre fichier **cloudbuild.yaml** une étape de création de release dans Cloud Deploy. Une Release représente les manifests générés par Skaffold pour chaque Target Cloud Deploy.

```
steps:
- id: Build the application
 name: 'gcr.io/k8s-skaffold/skaffold:slim'
 args: ['skaffold', 'build', '—file-output', 'artifacts.json', '—default-repo', 'europe-
west1-docker.pkg.dev/lgu-demos/spring-projects']
- id: Create a Cloud Deploy release
 name: 'gcr.io/cloud-builders/gcloud'
 args:
  [
   'deploy',
   'releases',
   'create',
   'run-release-${SHORT_SHA}',
   '—skaffold-version=skaffold_preview',
   '—region',
   'europe-west1',
   '—delivery-pipeline',
   'cloud-run-pipeline',
   '—build-artifacts',
   'artifacts.json'
  ]
```



Figure 5

Il est nécessaire lors de la création d'une Release d'ajouter plusieurs options comme le nom de la Release, la région où déployer l'application et le nom du pipeline à déclencher.

Ici, nous allons créer une Release nommée **run-release-${SHORT_SHA}** qui ajoutera automatiquement le dernier commit au nom de la Release. Cette Release va être créée en europe-west1 et nous allons déclencher le pipeline Cloud Deploy précédemment créé. Deux autres options ont été ajoutées : la version de Skaffold à utiliser pour pouvoir supporter Cloud Run et aussi l'utilisation du fichier artifacts.json contenant le nom complet de notre image dans la registry qui a été générée par Skaffold lors de l'étape précédente.

Comme nous l'avons vu précédemment, Cloud Build se base sur Skaffold afin de savoir quoi déployer. Il va falloir alors modifier notre fichier **skaffold.yaml** afin d'ajouter un nouveau profil qui représentera notre environnement de développement sur Cloud Run.

```
apiVersion: skaffold/v4beta2
kind: Config
build:
 artifacts:
  - image: spring-petclinic
   buildpacks:
    builder: "gcr.io/buildpacks/builder:v1"
    env:
     - GOOGLE_RUNTIME_VERSION=17
deploy:
 cloudrun: {}
profiles:
 - name: dev
  manifests:
   rawYaml:
    - run-dev.yaml
```

Ce nouveau profil pointe sur un fichier nommé **run-dev.yaml**. Ce fichier est un fichier représentant un Service Cloud Run.

La suite de notre dossier dans Programmez ! n°257.

Monitorer la santé des apps sans effort avec OpenTelemetry

L'Observabilité est un sujet en vogue. On désigne souvent ce terme comme étant le successeur de la supervision. Là où le monitoring traditionnel se basait uniquement sur le statut d'indicateurs connus comme l'utilisation processeur, la saturation mémoire ou le débit réseau, dans l'observabilité nous collectons tous les indicateurs possibles pour avoir une vision plus complète.



Brice DEKANY

Senior Observability Engineer chez VMware Aria

Adeptes des systèmes distribués Open Source je me suis spécialisé sur les sujets cloud Computing et Cybersécurité. J'aide aujourd'hui Dev et Ops à faire la paix en se retrouvant autour d'une pizza (toujours sans ananas)

vmware®

L'observabilité est un complément à la supervision d'infrastructure, via la capture de métriques et de traces internes à l'application, est essentielle pour connaître l'état de santé d'un service. Elle permet d'évaluer la qualité du service rendu, définir des axes d'améliorations et détecter des erreurs avant qu'elles ne deviennent des incidents.

À l'heure des micro-services et de leur prolifération, l'Observabilité devient **obligatoire** si on ne veut pas perdre la tête à chercher une aiguille dans une botte de foin au moindre incident ou dégradation de performance.

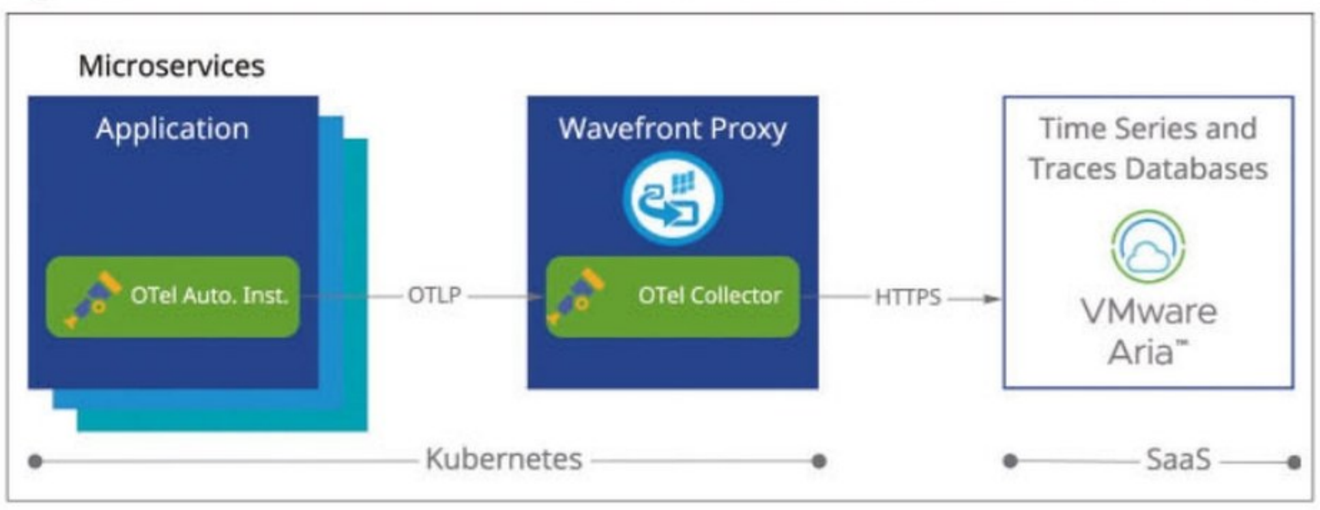
La bonne nouvelle c'est que grâce à OpenTelemetry, l'ajout de fonctions d'Observabilité à vos micro-services (ou des applications legacy) est automatique !

Pourquoi OpenTelemetry?

Pour faire de l'observabilité, nous avons besoins de métriques et de traces venant directement de l'application elle-même. Implémenter de nouvelles fonctions qui émettent ces signaux au sein du code va demander du temps et va générer beaucoup de questions, quelle nomenclature utiliser, quel protocole pour exporter les métriques et est-ce que cela va ralentir l'application ? OpenTelemetry répond à toutes ces questions pour nous. Ce projet open source soutenu par la CNCF va nous aider sur plusieurs points :

- C'est un standard qui est agnostique du langage de programmation. Les métriques, traces et logs générés dans une application micro-services utilisant plusieurs technos différentes n'est plus un problème.
- Le Registre OpenTelemetry, sorte de marketplace de bibliothèques, offre des implémentations pour quasiment tous les langages et surtout les bindings vers des middleware comme apache, redis ou rabbitMQ.
- C'est aussi un protocole de communication, le OpenTelemetry Protocol qui est un standard ouvert. Ce point est très important, car il permet d'instrumenter le code de son application sans se soucier du service qui va collecter et stocker les métriques et traces dans une time-series database.

Figure 1



Fonctionnement de OpenTelemetry ?

OpenTelemetry se découpe en plusieurs composants ce qui assure la flexibilité d'utilisation et la liberté quant au choix de providers de service. Le premier, celui au sein du code de l'application est l'Exporter. Il se base sur le SDK : c'est-à-dire les normes définies par Opentelemetry pour collecter et traiter les données. Il est possible d'implémenter un exporter soi-même dans le code source, ou d'utiliser les exporter créés par la communauté. L'auto-instrumentation est disponible pour plusieurs langages : Java, Python, nodeJS, .NET et Ruby. Le côté "Magic" de l'auto-instrumentation c'est qu'elle marche pour les fonctions Core de ces langages et aussi pour plusieurs bibliothèques externes gérants des middlewares.

Ce composant se veut le plus petit et le moins invasif possible. Son rôle est uniquement de générer des données et les pousser via le protocole OTLP à notre second composant : le collector. Le collector OpenTelemetry est un service qui va enregistrer les données des exporters. Le collector fonctionne en mode many-to-one afin de réduire la prolifération de services. Il écoute sur les ports 4317 en gRPC pour la majorité des langages ou en 4318 en http pour collecter les traces web, type JavaScript. L'intérêt est que ce collector générique peut ensuite stocker les données dans une base de type time-series database qu'elle soit en local ou un service managé tel que Aria Operations for Applications.

Architecture et Automatisation

Vient maintenant la partie la plus excitante d' OpenTelemetry, celle où l'on ne fait rien, car tout est automatisé. **Figure 1**

Je vais utiliser une plateforme d'orchestration Kubernetes et profiter des operators pour installer et automatiser le processus d'export et de collecte des métriques de micro-services.

Les operators sont des extensions de l'API de kubernetes qui facilitent grandement la gestion et le cycle de vie de composants tierces.

L'operator Kubernetes d'OpenTelemetry permet d'injecter les exporters dans une application déjà containérisée ! Puis d'envoyer les traces et les métriques via OTLP au Collecor voulu, dans notre cas Aria Operations for Applications.

En plus de ne pas avoir à toucher le code de notre application, l'injection de l'auto-instrumentation permet d'observer des applications dont nous ne maîtrisons pas. Même chose

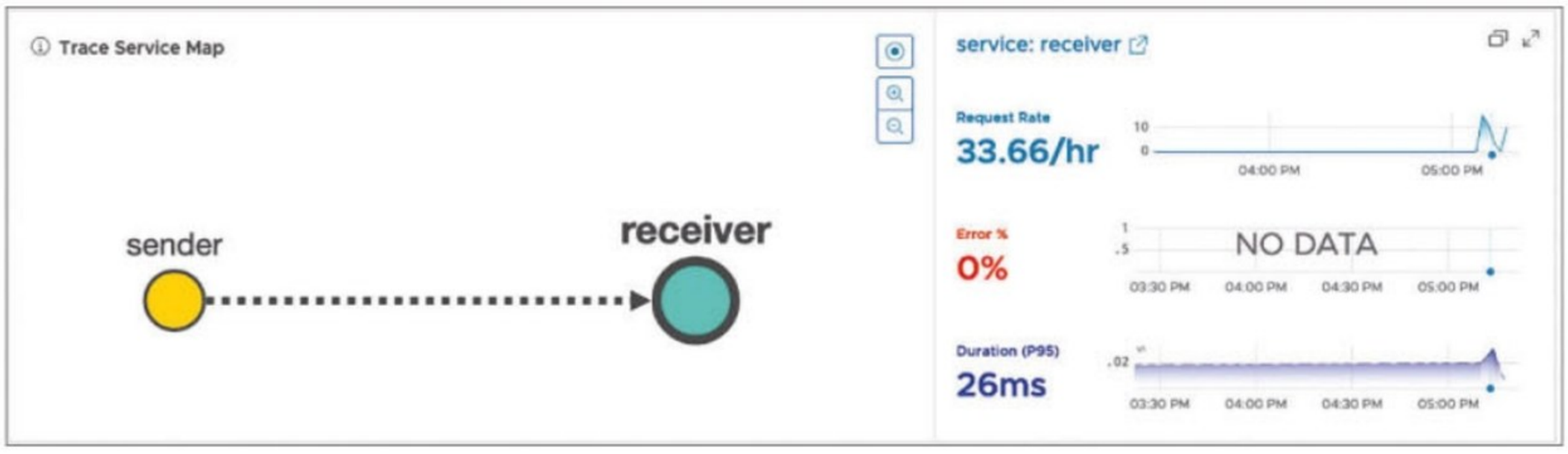


Figure 2

pour la partie Collector, je vais utiliser l'operator Wavefront pour kubernetes. Cet operator va collecter les métriques propres à Kubernetes (pod, namespace, nodes) et va déployer un proxy compatible avec OTLP pour capter métriques, traces et logs envoyer par l'exporter. Dernière phase de l'automatisation, un compte sur Aria Operation for Applications notre solution SaaS de timeseries database et de corrélation de données pour visualiser toutes nos données.

En plus d'offrir une solution de visualisation, Aria for Applications utilise les traces pour générer un graph de dépendances et les RED Metrics utilisés par les SRE et platform Engineer popularisés par Google (Rate Error Duration).

Figure 2

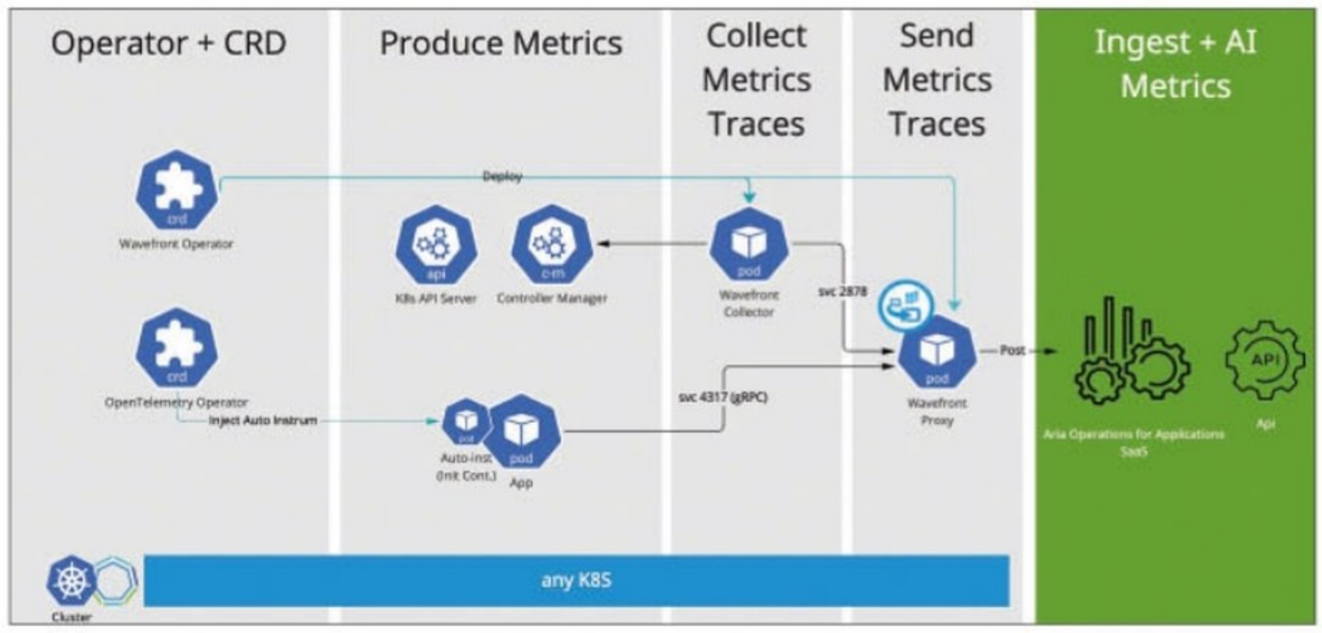


Figure 3

Installation des composants dans un environnement Kubernetes **Figure 3**

Mettons tout cela en ouvre avec une application Java qui utilise RabbitMQ pour publier et lire des messages. Les métriques et appels de fonctions internes sont envoyés sur Aria for Applications qui crée automatiquement une visualisation de l'architecture et un tableau de bord basé sur les indicateurs les plus importants pour suivre la qualité de service.

Prérequis

- Un environnement Kubernetes >= 1.19
- Un compte Trial sur VMware Aria Operations for Applications (pour l'API KEY et l'URL de votre tenant)

[Get Trial Code] https://tanzu.vmware.com/observability-trial)

Installation du Wavefront Proxy (OTel Collector Service)

Le Wavefront Proxy va collecter des informations sur l'environnement Kubernetes et servir de Collector OpenTelemetry. Pour l'installer, il suffit d'appliquer le manifeste de l'operator au format YAML.

```
kubectl apply -f https://raw.githubusercontent.com/wavefrontHQ/wavefront-operator-for-kubernetes/v2.1.0/deploy/kubernetes/wavefront-operator.yaml
kubectl wait pods -l app.kubernetes.io/component=controller-manager -n observability-system —for=condition=Ready
```

Une fois installée, nous allons déclarer notre clef d'API récupérer sur la solution SaaS Aria for Applications (https://longboard.wavefront.com/) et configurer l'objet Wavefront pour qu'il pointe vers notre tenant sans oublier d'activer la compatibilité OTLP.

```
export YOUR_WAVEFRONT_TOKEN=aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeee
kubectl create -n observability-system secret generic wavefront-secret —from-literal token=${YOUR_WAVEFRONT_TOKEN}
```

```
kubectl apply -f - <<EOF
apiVersion: wavefront.com/v1alpha1
kind: Wavefront
metadata:
 name: wavefront
 namespace: observability-system
spec:
 clusterName: dbrice-gke
 wavefrontUrl: https://longboard.wavefront.com/
 dataCollection:
  metrics:
   enable: true
 dataExport:
  wavefrontProxy:
   enable: true
   otlp:
    grpcPort: 4317
    httpPort: 4318
    resourceAttrsOnMetricsIncluded: true
EOF
```

Installation de l'operator OpenTelemetry

Même process pour l'operator OpenTelemetry, nous allons appliquer le manifest au format YAML et configurer l'auto-injection qui va nous permettre d'observer nos applications sans modifier leur code. OpenTelemetry a besoin de cert-manager pour chiffrer les communications.

```
kubectl apply -f https://github.com/cert-manager/cert-manager/releases/download/v1.8.0/cert-manager.yaml
kubectl wait pods -l app=webhook -n cert-manager —for=condition=Ready
kubectl apply -f https://github.com/open-telemetry/opentelemetry-
```

```
operator/releases/download/v0.67.0/opentelemetry-operator.yaml
kubectl wait pods -l control-plane=controller-manager -n opentelemetry
-operator-system —for=condition=Ready
```

Configuration de l'Instrumentation pour auto-injection

Nous définissons un objet Instrumentation au niveau de Kubernetes. Tout pod qui aura une annotation spécifique se verra injecter la bonne bibliothèque directement au niveau du container, sans avoir à toucher au code de l'application.

Exemple d'annotation :

```
instrumentation.opentelemetry.io/inject-java: "namespace/ontelinst-name"
```

Dans cette section nous spécifions que le collector est notre Wavefront proxy installé précédemment.

```
kubectl apply -f - <<EOF
apiVersion: opentelemetry.io/v1alpha1
kind: Instrumentation
metadata:
 name: my-instrumentation
spec:
 exporter:
  endpoint: http://wavefront-proxy.observability-system:4317
 propagators:
  - tracecontext
  - baggage
  - b3
 sampler:
  type: parentbased_traceidratio
EOF
```

Démonstration avec une application Java Spring et RabbitMQ

Pour la partie démo, je vous propose d'utiliser une application assez simple, deux services qui échangent des données via RabbitMQ. Nos deux services Java déjà containerisés vont se voir injecter le bon exporter via notre operator OpenTelemetry.

Pré requis Deployer Rabbit MQ

Je déploie un serveur RabbitMQ via un chart Helm pour rester dans mon optique d'automatiser au maximum. Notez que j'active le drapeau « metrics » lors du déploiement. Ainsi notre operator Wavefront va pouvoir collecter des informations propres à RabbitMQ (taille de file, nombre de clients, topics,…)

```
helm repo add bitnami https://charts.bitnami.com/bitnami && helm
repo update
```

```
helm install rabbitserver bitnami/rabbitmq \
—set persistence.enabled=false \
—set metrics.enabled=true \
—set auth.username=tutorial \
—set auth.password=tutorial \
—set nameOverride=rabbitserver
```

Déploiement des deux services Java

Enfin il nous reste à lancer nos deux services sous forme de pod. L'un sera le sender, l'autre le receiver. Il y aura également l'annotation pour que l'operator OpenTelemetry injecte l'exporter Java.

```
instrumentation.opentelemetry.io/inject-java: "default/my-instrumentation"
```

```
kubectl run sender —image docker.io/bdekany/rabbitmq-tutorials:spring
-amqp —annotations=instrumentation.opentelemetry.io/inject-java
="default/my-instrumentation" —env=OTEL_RESOURCE_ATTRIBUTES
="application=rabbitApp" —command — java -jar rabbitmq-tutorials
.jar —spring.profiles.active=hello-world,sender,remote

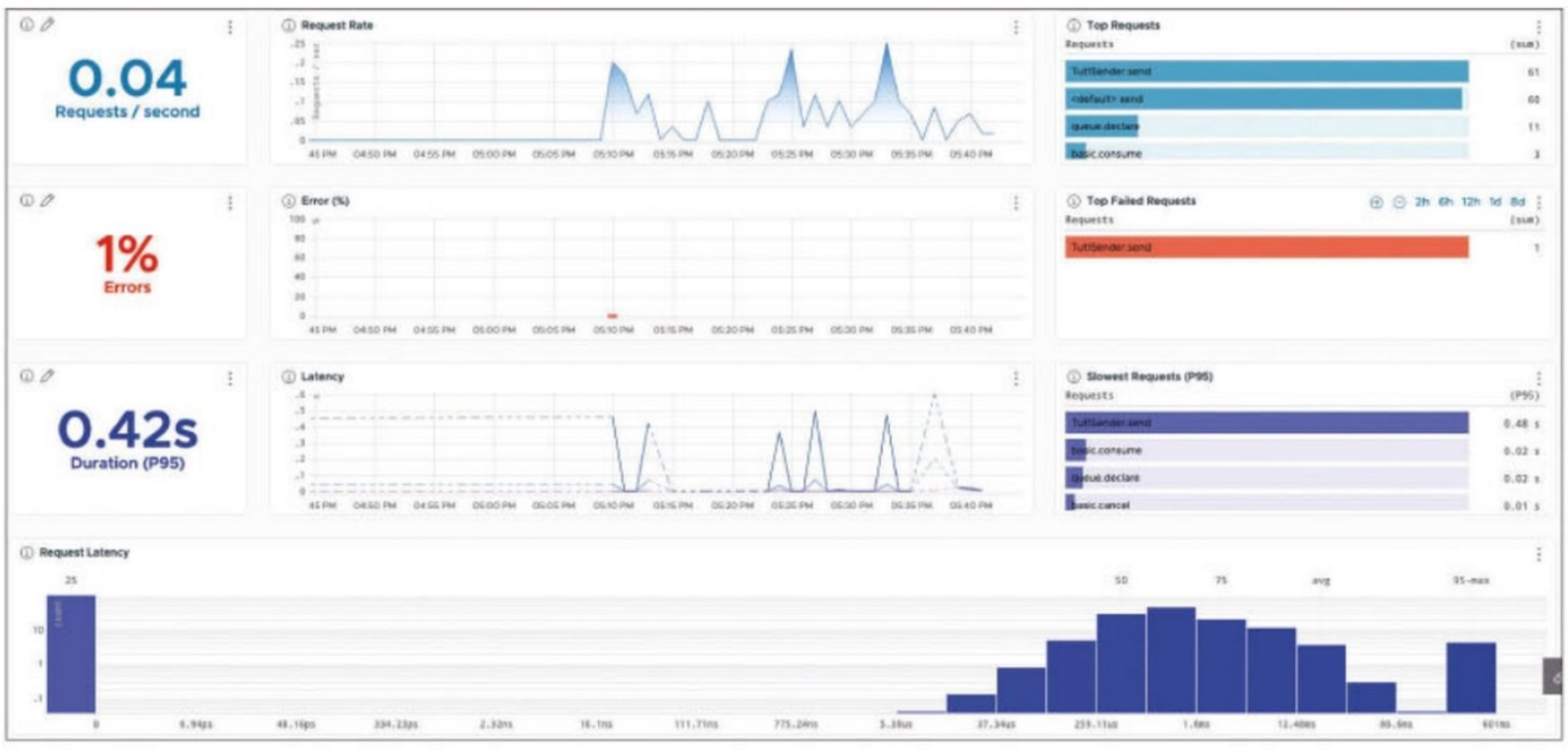
kubectl run receiver —image docker.io/bdekany/rabbitmq-tutorials:spring
-amqp —annotations=instrumentation.opentelemetry.io/inject-java
="default/my-instrumentation" —env=OTEL_RESOURCE_ATTRIBUTES
="application=rabbitApp" —command — java -jar rabbitmq-tutorials
.jar —spring.profiles.active=hello-world,receiver,remote
```

Connectez-vous sur la solution SaaS pour découvrir les traces et Red Metrics auto générées. (https://longboard.wavefront.com/)

Figure 4

À vous de jouer !

Figure 4



Développer un module Azure IoT Edge en .NET C#



Kevin BEAUGRAND

Craftsman developer - MVP Internet of Things – CGI

Avec plus de 10 ans d'expérience dans l'analyse et la conception de logiciels dans différents secteurs de l'industrie je suis devenu un expert dans la mise en place de solutions basées sur les technologies .NET Azure Cloud et IoT.

Azure IoT Edge est une solution s'appuyant sur Azure IoT Hub pour déployer et gérer des charges de travail sur des appareils physiques connectés au réseau d'IoT locales. Il est notamment possible d'exécuter des traitements métiers sur la base des données IoT au plus proche des appareils et objets. Nous allons voir ensemble comment il est possible de développer soi-même un module pour Azure IoT Edge permettant de capter de la donnée depuis un automate programmable et l'envoyer dans le Cloud.

Les automates de productions

Les automates programmables (PLC dans le jargon), sont des ordinateurs utilisés dans l'industrie pour réaliser les opérations d'automatisation de la production. En clair, ils sont les cerveaux des machines de production d'une usine et réalisent la logique et la communication avec les éléments mécaniques la machine, la rendant ainsi vivante. Être capable de communiquer avec nous donne la possibilité de piloter la machine de production (lui spécifier quand démarrer un cycle, quand l'arrêter par exemple), la paramétrer, mais également récupérer des informations permettant de diagnostiquer le fonctionnement de la machine.

Prérequis

Pour réaliser ce tutoriel, vous aurez besoin de logiciels et de bibliothèques : Visual Studio Code (avec les extensions C# Azure IoT Edge et Azure IoT Hub), .NET SDK. De plus Azure IoT Edge s'appuie sur Docker pour exécuter les modules, vous devrez donc également installer un runtime d'exécution de conteneurs Docker Linux sur votre machine de développement.

De plus, pour pouvoir tester la solution sur votre poste, vous devrez posséder un tenant Azure ainsi qu'une instance Azure IoT Hub (Azure IoT Hub est également disponible gratuitement avec un compte Azure free).

Préparation de la solution Azure IoT Edge

Une fois les prérequis installés et configurés, nous pouvons maintenant démarrer et créer une nouvelle solution Azure IoT Edge à l'aide de VS Code.

À l'aide de la palette de commande de Visual Studio Code (Touche **F1**), entrez la commande **Azure IoT Edge : New IoT Edge solution**. Vous serez invités à sélectionner un répertoire, le nom de la solution, le template du module à utiliser (sélectionnez **C# Module**), le nom du module et enfin l'URI de votre container registry (vous pourrez laisser la valeur par défaut).

Anatomie de la solution

Une solution Azure IoT Edge est constituée à minima des éléments suivants :

- **deployment.template.json** : template de déploiement IoT Edge utilisé par le development toolkit pour exécuter la solution sur votre poste de développement.
- **modules/<votre module>** : le répertoire de votre module IoT Edge où l'on retrouve les fichiers suivants :
  - **Program.cs** : fichier contenant les instructions de niveau supérieur (Top-Level statements) de l'application. Par defaut, le programme démarre une tâche background présente dans le **ModuleBackgroundService.**
  - **ModuleBackgroundService.cs** : classe de service exécutant le code de votre module. Elle est instanciée et exécutée lors du démarrage du module.
  - **module.json** : fichier descripteur de votre module.
  - **Dockerfile.*** : ensemble des fichiers d'instructions de création des images Docker en fonction des architectures processeurs où le module sera exécuté.

Le template du module IoT Edge propose d'ingérer et traiter des évènements dans le module. Notre projet doit pour sa part capter des données de notre PLC et les envoyer dans le cloud, nous allons donc épurer le code de notre module :

```
internal class ModuleBackgroundService : BackgroundService
{
  private int _counter;
  private ModuleClient? _moduleClient;
  private CancellationToken _cancellationToken;
  private readonly ILogger<ModuleBackgroundService> _logger;

  public ModuleBackgroundService(ILogger<ModuleBackgroundService
> logger) => _logger = logger;

  protected override async Task ExecuteAsync(CancellationToken cancellation
Token)
  {
    _cancellationToken = cancellationToken;
    MqttTransportSettings mqttSetting = new(TransportType.Mqtt_Tcp
_Only);
    ITransportSettings[] settings = { mqttSetting };

    // Open a connection to the Edge runtime
    _moduleClient = await ModuleClient.CreateFromEnvironmentAsync
(settings);
```

```
    // Reconnect is not implemented because we'll let docker restart the
process when the connection is lost
    _moduleClient.SetConnectionStatusChangesHandler((status, reason) =>
      _logger.LogWarning("Connection changed: Status: {status} Reason:
{reason}", status, reason));

    await _moduleClient.OpenAsync(cancellationToken);

    _logger.LogInformation("IoT Hub module client initialized.");
  }
}
```

Maintenant que notre projet est prêt, nous pouvons nous attaquer au code permettant de nous connecter à l'automate de production et envoyer ses données.
Commençons par installer les dépendances NuGet nécessaires à notre développement :

```
dotnet add package libplctag
```

Note : *La librairie libplctag est une librairie OpenSource permettant la communication avec les automates programmables. Il s'agit d'une librairie très simple d'utilisation et qui permet de communiquer avec une grande variété de PLC de type ControlLogix, très répandu dans le monde industriel.*

Modifications des droits dans le container

La librairie libplctag, nécessite des droits de lecture et d'écriture dans le répertoire de l'application dans le conteneur où nous allons exécuter le module. Si nous ne faisons rien, alors nous aurons le malheur de tomber sur ce message d'erreur lors de l'exécution de l'application :

```
Unhandled exception. System.UnauthorizedAccessException: Access to the path
'/app/libplctag.so' is denied.
```

Pour ne pas rencontrer cette erreur, je vous propose de modifier les fichiers Dockerfile de notre module pour changer le propriétaire du répertoire de l'application dans l'image du conteneur Docker. Modifions les fichiers Dockerfile pour modifier ces droits :

```
WORKDIR /app
COPY --from=build-env /app/out ./

RUN useradd -ms /bin/bash moduleuser && chown -R moduleuser /app
USER moduleuser

ENTRYPOINT ["dotnet", "PLCModule.dll"]
```

Les propriétés de module IoT Edge

Tout comme un IoT Device, la plateforme Azure IoT Hub permet de gérer des paramètres de fonctionnement des modules IoT Edge. Dans le monde Azure IoT, cela s'appelle des propriétés, nous avons à notre disposition 2 types de propriétés : les **propriétés désirées** (celles que nous fournissons à notre appareil pour le configurer, les **propriétés reportées** (celles que l'appareil nous notifie posséder).

Note : *Dans Azure IoT Hub, l'ensemble des paramètres des appareils, IoT Edge, et modules sont enregistrées sous la forme d'un jumeau numérique (Digital Twin). Nos propriétés seront quant à elles stockées dans le Module Twin dans une représentation Json.*

```
using libplctag;

public class TagDefinition
{
  public string Name { get; set; }

  public string Gateway { get; set; }

  public string Path { get; set; }

  public string TagName { get; set; }

  public int PollingInterval { get; set; }

  public PlcType PlcType { get; set; } = PlcType.ControlLogix;
}
```

Nous allons dans notre tutoriel utiliser les propriétés désirées pour configurer le module et spécifier quels sont les tags de notre PLC que l'on veut acquérir et à quelle fréquence.

Passons au code de notre module

C'est parti pour le cœur de notre atelier ! Commençons par créer la fonction qui aura la charge de se connecter à l'automate et d'envoyer un évènement dans IoT Edge :

```
  private async Task ReadAndSendTagValueAsync(TagDefinition tag)
{
    using var plcTag = new Tag<DintPlcMapper, int>();

    plcTag.Name = tag.TagName;
    plcTag.Gateway = tag.Gateway;
    plcTag.Path = tag.Path;
    plcTag.PlcType = tag.PlcType;
    plcTag.Protocol = Protocol.ab_eip;
    plcTag.ReadCacheMillisecondDuration = 0;

    this._logger.LogTrace("Polling tag {0}", tag.TagName);
    plcTag.Read();

    if (plcTag.GetStatus() != Status.Ok)
    {
      this._logger.LogWarning($"Tag {tag.TagName} is not OK - {plcTag
.GetStatus()}");
      return;
    }

    var jsonTagValue = JsonConvert.SerializeObject(plcTag.Value);
   var jsonEvent = JObject.Parse($"{{\"{tag.Name}\":{jsonTagValue}}}");
    var jsonData = jsonEvent.ToString();

   this._logger.LogTrace("Sending message for tag {0} with value: {1}",
tag.TagName, jsonData);

    var message = new Message(Encoding.UTF8.GetBytes(jsonData));
```

```
    await this._moduleClient.SendEventAsync("tag", message);

    this._logger.LogTrace("Message queued for tag {0}", tag.TagName);
  }
```

Cette fonction a donc la charge de se connecter au PLC depuis la définition que nous lui fournissons, lire la donnée et la sérialiser en JSON pour que nous puissions ensuite l'envoyer dans le broker EdgeHub puis le traiter dans la plateforme Azure IoT (que ce soit en Edge ou dans le Cloud).
Nous devons maintenant créer la fonction qui effectuera le polling de manière régulière :

Code complet sur programmez.com & github

Nous savons désormais déclencher de manière régulière les lectures et envois des données, nous pouvons récupérer le paramétrage de notre module et lancer les polling sur cette configuration.

Code complet sur programmez.com & github

Il ne nous reste plus maintenant qu'à orchestrer cela lors du démarrage du module, ajoutons les lignes suivantes à la fin de la fonction **ExecuteAsync** :

```
    var twin = await this._moduleClient.GetTwinAsync();
    RegisterTags(twin.Properties.Desired);
```

Déploiement du module

Voilà, nous sommes prêts à tester notre module. Pour pouvoir le lancer sur notre poste de développement, nous allons devoir modifier le template de déploiement IoT Edge (**deployment.template.json**). C'est ici que l'on définie l'ensemble des modules devant s'exécuter sur IoT Edge et leur configuration. Par défaut VSCode a déjà ajouté une référence notre notre module dans la configuration du Edge Agent (cela permet de spécifier à l'agent edge de déployer notre module). Nous devons cependant modifier le template pour ajouter la configuration de notre module (les propriétés désirées).

```
{
 "$schema-template": "4.0.0",
 "modulesContent": {
  "$edgeAgent": {
    ...
  },
  "PLCModule": {
   "properties.desired": {
    "tags": [
     {
      "name": "Awesome_tag",
      "gateway": "192.168.1.24",
      "path": "1,0",
      "tagName": "SOME_TAG_NAME",
      "pollingInterval": "1000"
     }
    ]
   }
 },
 "$edgeHub": {
    ...
  }
 }
}
```

Test de la solution

Pour tester la solution, nous devons configurer notre machine pour se connecter à notre IoT Hub. VSCode a une fois de plus des commandes disponibles nous facilitant la configuration. Entrez successivement les commandes suivantes :

- Azure IoT Hub: Select IoT Hub
- Azure IoT Edge: Create IoT Edge Device
- Azure IoT Edge: Setup IoT Edge Simulator

Après avoir renseigné les informations nécessaires, nous avons maintenant la possibilité de lancer la solution sur notre poste de développement : **Azure IoT Edge : Build and Run IoT Edge Solution in Simulator**.

```
PLCModule    |<6>PLCModule.ModuleBackgroundService[0] Registering tags
PLCModule    |<7>PLCModule.ModuleBackgroundService[0] Tag: SOME_TAG_NAME, PollingInterval: 1000
PLCModule    |<7>PLCModule.ModuleBackgroundService[0] Starting polling for tag SOME_TAG_NAME
PLCModule    |<7>PLCModule.ModuleBackgroundService[0] Polling tag SOME_TAG_NAME
PLCModule    |<7>PLCModule.ModuleBackgroundService[0] Sending message for tag SOME_TAG_NAME with value: {  "Awesome_tag": 54 }
PLCModule    |<7>PLCModule.ModuleBackgroundService[0] Message queued for tag SOME_TAG_NAME
```

Regardons maintenant les évènements que IoT Hub reçoit. Avec VSCode entrons la commande **Azure IoT Hub: Start Monitoring Built-in Event Endpoint**. Nous pouvons maintenant observer les évènements reçus dans le cloud :

```
[IoTHubMonitor] Start monitoring message arrived in built-in endpoint for all devices ...
[IoTHubMonitor] Created partition receiver [0] for consumerGroup [$Default]
[IoTHubMonitor] Created partition receiver [1] for consumerGroup [$Default]
[IoTHubMonitor] [6:13:55 PM] Message received from [dev-kbe/PLCModule]:
{
 "Awesome_tag": 86
}
[IoTHubMonitor] [6:13:56 PM] Message received from [dev-kbe/PLCModule]:
{
 "Awesome_tag": 42
}
```

Conclusion

Nous avons vu comment développer et tester un module fonctionnant sous Azure IoT Edge. Ce module permet de communiquer directement avec un automate programmable et lire ses tags avant de les renvoyer dans EdgeHub. Une fois les données collectées et envoyées dans le broker, nous pouvons facilement plugger d'autres modules qui bénéficieront de ces évènements pour effectuer des traitements que ce soit local ou cloud…

Google Cloud Armor et Cloud CDN : configuration et tips

Google Cloud Armor et Google Cloud CDN sont deux services de Google dans le cadre de son offre Cloud qui permettent d'améliorer la sécurité et les performances d'un site ou d'une application web. Dans cet article, nous verrons comment configurer ces deux solutions afin d'optimiser la sécurité et les performances de votre application web. Nous listerons également les fonctionnalités avancées et quelques conseils pour l'optimisation de ses deux outils.



Tanguy BAUDRIN

Staff Engineer Cloud et Google Cloud Authorized Trainer chez SFEIR

Véritable passionné d'informatique depuis toujours, et après plus de 10 années dans le développement logiciel dans diverses ESN, j'ai rejoint Sfeir il y a 5 ans. Depuis, mon terrain de jeu est devenu l'infrastructure Cloud, avec des compétences particulières autour du DevOps, de l'automatisation et de Kubernetes.



Nidal EL HADDAD

Ingénieure DevOps chez SFEIR

Je suis ingénieure DevOps, chez SFEIR depuis 1 an et demi. J'ai également plus de 7 ans d'expérience autour de l'observabilité, l'automatisation et l'amélioration continue. Je suis passionnée par la mise en place de solutions automatisées, la conteneurisation ainsi que la sécurité informatique.

Fondamentaux de Google Cloud CDN

Google Cloud CDN (Content Delivery Network) est un service de diffusion de contenu qui accélère la livraison de votre contenu aux utilisateurs finaux. Il utilise plus de 134 points de présence répartis dans le monde entier pour mettre à disposition vos fichiers statiques (images, vidéo, fichiers CSS, etc.) auprès de vos utilisateurs en utilisant le serveur le plus proche géographiquement. Cela permet de réduire les temps de chargement de votre site web et d'améliorer l'expérience utilisateur. Google Cloud CDN est accessible à tous les utilisateurs de Google HTTP (S) Load Balancer et offre de nombreuses fonctionnalités, telles que HTTP/2 et HTTP/3 basé sur le protocole QUIC de l'IETF (Internet Engineering Task Force), Anycast, TLS version 1.3, logging et monitoring. En outre, il permet de purger le cache instantanément.

Fondamentaux de Google Cloud Armor

Google Cloud Armor est un service de sécurité réseau qui protège contre les attaques de type DDoS (Distributed Denial of Service) et les attaques d'application ciblées. Il utilise des règles de sécurité qui analysent en temps réel le trafic entrant et bloquent les requêtes malveillantes avant qu'elles n'atteignent le service. Google utilise Cloud Armor pour protéger sa propre infrastructure mondiale (Google Search, Gmail et YouTube) contre les attaques de type DDoS. En effet, lors de la plus grosse cyberattaque de l'histoire sur son cloud, Cloud Armor a été capable de détecter cette attaque très tôt grâce à sa fonctionnalité de protection adaptative, qui a agi comme un pare-feu et a permis le déclenchement de mécanismes de protection. Cette attaque a impliqué 46 millions de requêtes par seconde, mais grâce à Cloud Armor, Google a été en mesure de la contrer efficacement.

QUELLE ARCHITECTURE ?

Ci-dessous une vision de l'architecture mise en place lors de l'utilisation de Google Cloud Armor et de Google Cloud CDN

Figure A

Les requêtes utilisateurs sont acheminées au travers du réseau Internet vers le Google Front end (GFE) le plus proche du client.

Le GFE va être capable de consulter directement le cache du CDN et pouvoir répondre directement à l'utilisateur sans avoir besoin de contacter votre infrastructure. Permettant donc une optimisation de l'utilisation de vos ressources cloud En revanche, lorsque le contenu demandé par l'utilisateur n'est pas disponible dans le cache, le GFE transfère la requête au load balancer global. Ce dernier transfère ensuite la requête au backend service le plus proche

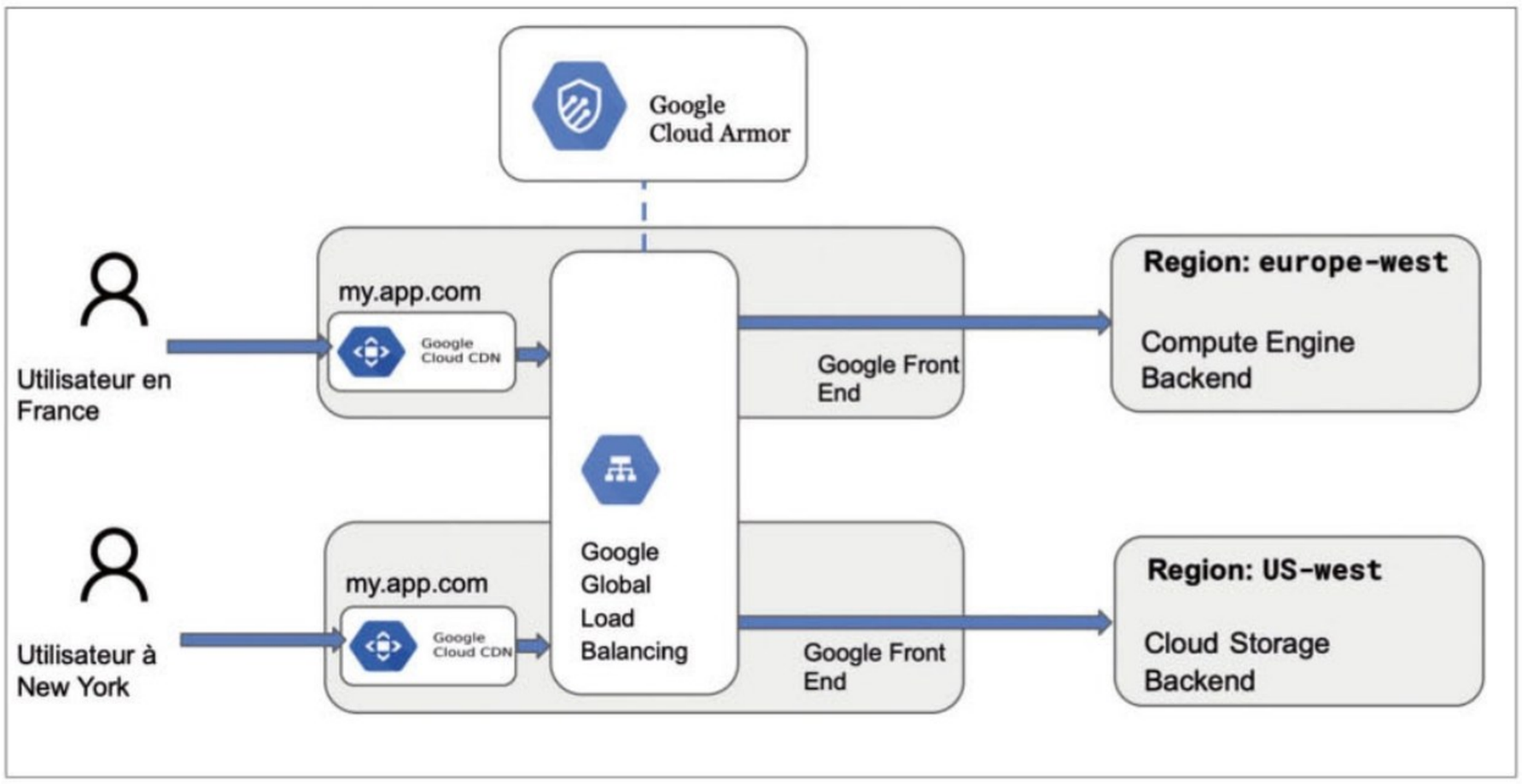


Figure A

géographiquement de l'utilisateur. Ces backends services peuvent être un ou plusieurs de ses éléments sur Google Cloud: Groupe de machines virtuelles, Cluster GKE, App Engine, Cloud Run, Cloud Function et bucket GCS et enfin d'autres cibles en utilisant les network endpoint group de Google.
Cloud armor est placé au niveau du load balancer, et va permettre d'analyser le trafic entrant avant d'être renvoyé à vos backends.

MISE EN PLACE AVEC TERRAFORM

L'ensemble du code terraform est disponible sur le dépôt git: https://github.com/TanguyBaudrin/terraform-googlecloud-lb-cloud-armor-cdn
Le dépôt git contient tout le code pour démarrer depuis un projet vide, dans la suite de l'article nous nous concentrerons sur les éléments nécessaires à Cloud Armor et Cloud CDN.

Déploiement du load balancer

Code terraform utilisant le module de load balancer Google

```
module "gce-lb-http" {
 source          = "GoogleCloudPlatform/lb-http/google"
 version         = "6.3.0"
 name            = "https"
 [...]
 backends = {
  this = {
   [...]
   enable_cdn              = true
   security_policy         = google_compute_security_policy.this.self_link
   [...]
 }
}
```

Dans cet exemple, nous utilisons le module terraform officiel de Google qui facilite grandement la mise en place d'un load balancer.

Pour la partie Cloud Armor et Cloud CDN, ce qui nous intéresse ce sont les deux paramètres enable_cdn et security_policy. Le premier permet simplement d'activer Cloud CDN, le second va nous permettre d'activer Cloud Armor en appliquant une règle de security policy.

A noter, les règles Cloud Armor sont liées aux différents backend services, vous permettant d'avoir des règles différentes en fonction de vos différents types de backend. Ou au contraire réutiliser les mêmes règles pour l'ensemble de vos backend services.

MISE EN PLACE DE CLOUD ARMOR

Pour rappel, Cloud Armor est la solution de Google pour se protéger contre les attaques par déni de service (DDoS) et les attaques d'application ciblées.

Filtrer les requêtes utilisateurs par ip "statique"

La configuration de vos règles Cloud Armor utilise la ressource google_compute_security_policy, une seule ressource va vous permettre de configurer l'ensemble des règles à appliquer. Voici un exemple de mise en place d'une règle de blocage sur certaines plages d'ip sur Cloud Armor.

```
resource "google_compute_security_policy" "this" {
 name = "my-policy"

 rule {
  action = "deny(403)"
  priority = "1000"
  match {
   versioned_expr = "SRC_IPS_V1"
   config {
    src_ip_ranges = ["9.9.9.0/24"]
   }
  }
  description = "Deny access to IPs in 9.9.9.0/24"
 }

 rule {
  action = "allow"
  priority = "2147483647"
  match {
   versioned_expr = "SRC_IPS_V1"
   config {
    src_ip_ranges = ["*"]
   }
  }
  description = "default rule"
 }
}
```

Dans cet exemple nous créons deux règles, la première qui va interdire les ip sources dans le range "9.9.9.0/24" avec une priorité de 1000, et la seconde qui va autoriser toutes les ip avec une priorité de 2147483647.
Une valeur de priorité plus petite indique une règle qui sera appliquée en priorité. Dans notre cas, la règle interdisant les ip en "9.9.9.0/24" étant plus "petite" elle sera prioritaire sur l'autorisation de toutes les ips.

Attention, une limitation sur cette fonctionnalité, vous ne pouvez pas déclarer plus de 10 adresses IP pour une seule règle. Si vous souhaitez bloquer un grand nombre d'adresses IP, il vous faudra donc les déclarer dans des règles différentes par paquet de 10.

Filtrer des listes d'ip connues comme malveillantes ou inappropriées

Cloud armor propose une fonctionnalité de "Threat intelligence" qui est incluse dans l'offre Managed Protection Plus et qui vous permet, au lieu de bloquer des listes d'ips manuellement, d'utiliser des listes d'adresses IP maintenus par Google.
Ces IPs sont globales et vous n'aurez pas la possibilité d'y

ajouter vos propres listes, mais elle reprennent des IP connues dans différentes catégories :

- Liste des nœuds de sortie du réseau Tor
- Liste des ip reconnues comme étant malveillantes
- Liste des robots scanneurs des moteurs de recherche
- Liste IP de proxies de Cloudflare, Fastly, Imperva
- Liste IP des principaux cloud providers, AWS, Azure, Google

Dans notre cas nous allons ajouter à notre code terraform, la liste des ip connues comme étant malveillantes, pour ce faire il suffit d'ajouter une règle à notre security policy :

```
rule {
 action    = "deny(403)"
 priority  = "1001"
 description = "Matches IP addresses known to attack web applications"
 match {
  expr {
   expression = "evaluateThreatIntelligence('iplist-known-malicious-ips')"
  }
 }
 preview = false
}
```

Dans cet exemple, nous créons une règle avec une priorité 1001 qui bloque les IP qui sont réputées malicieux selon Google.

Filtrer les requêtes identifiées comme malveillantes

Cloud armor propose également un ensemble de règles qui pourront vous protéger des requêtes malveillantes classiques de type injection sql, cve, etc.

La liste est importante, et l'ensemble de ses règles est disponible ici directement dans la documentation de Google : https://cloud.google.com/armor/docs/waf-rules

On pourra noter un exemple qui a fait couler beaucoup d'encre et de sueurs dans les équipes IT, avec la faille log4shell. Sur Cloud Armor, il est rapide est facile de bloquer ces requêtes au niveau du load balancer en activant la règle cve-canary.

Le choix de ces règles va dépendre du type d'application hébergées sur vos services de backend, en effet si votre application est une application Java, les règles ne seront pas les mêmes que s'il s'agit d'une application NodeJS ou PHP.

Dans notre cas,nous souhaitons ajouter une règle de protection de type "Remote code execution", pour ce faire nous utiliserons la règle rce-v33-stable :

```
rule {
 action    = "deny(403)"
 priority  = "1003"
 description = "Remote code execution requests"
 match {
  expr {
   expression = "evaluatePreconfiguredExpr('rce-v33-stable')"
  }
 }
}
```

Ces règles peuvent être encore affinées pour extraire certaines données de l'analyse faite par Cloud Armor, si vous constatez de faux positifs vous pourrez ainsi corriger le tir et vous assurer que les requêtes légitimes ne sont pas bloquées. De même vous pouvez affiner les différentes signatures, pour exclure, ou ajouter des signatures de requêtes qui ne seraient pas disponible de base dans votre règle.

Limiter le nombre de requêtes possibles pour un client

Cloud armor propose également une fonctionnalité de limitation de nombre de requêtes qui permet de réduire plus facilement le trafic vers les ressources backend en fonction du volume de demandes. Permettant ainsi de protéger vos ressources backend d'attaques de type DDoS.

```
rule {
 action    = "deny(403)"
 priority  = "1002"
 description = "rate-based-ban path '/' (10 rpm throttle 20 rpm ban )"
 match {
  expr {
   expression = "request.path.matches('/')"
  }
 }
 preview = true
 rate_limit_options {
  enforce_on_key = "IP"
  exceed_action = "deny(429)"
  rate_limit_threshold {
   count      = 10
   interval_sec = 60
  }
  ban_duration_sec = 3600
  ban_threshold {
   count      = 20
   interval_sec = 60
  }

 }
}
```

Dans cette règle, nous appliquons une limitation de nombre requêtes sur le path "/" en se basant sur l'ip du client. Vous pouvez également définir des règles avec des conditions de correspondance se basant soit sur des listes d'adresses/de plages IP ou des expressions comprenant plusieurs sous-expressions pour mettre en correspondance divers attributs d'une requête entrante tel que le code de la région, le Schéma d'URL HTTP, l'asn...

Dans cet exemple, si le nombre de requêtes sur un intervalle de 1 min est >= 10 et < 20 on renvoie une réponse 429 (au choix parmi 403, 404, 429 et 502). Sinon si le nombre de requêtes sur un intervalle de 1 min est >= 20 on renvoie une 403 pendant 1h.

Attention avec ce genre de règles, vous pouvez bloquer les requêtes provenant de votre réseau d'entreprise qui n'a

qu'un nombre limité d'adresses ip Internet pour un nombre important d'utilisateurs. Pensez alors à avoir une requête de priorité plus faible autorisant le trafic légitime de ces sources.

Cloud Armor permet de prévisualiser les effets d'une règle avant de l'appliquer avec le mode preview=true. Cela permet de tester la règle et d'éviter de bloquer le trafic légitime.
Le détail de la requête est consigné dans Cloud Logging (il est recommandé d'activer la journalisation détaillée, à l'aide de l'option —log-level=VERBOSE, afin de déterminer pourquoi la règle a été déclenchée)

Gestion des bots

Cloud Armor permet également de protéger vos services de backend pour vous assurer que seuls des utilisateurs humains pourront accéder à vos services.
Ce service dépend de l'utilisation de la solution reCAPTCHA entreprise, qui est un autre produit de la suite de Google. Pour activer ce service en utilisant votre code terraform, voici un exemple d'implémentation:

```
rule {
 action      = "redirect"
 description = "redirect to recaptcha"
 match {
  expr {
   expression = "request.path.contains('/login')"
  }
 }
 preview  = true
 priority = 1004
 redirectOptions {
  type = "GOOGLE_RECAPTCHA"
 }
}
```

Dans cet exemple La requête de l'utilisateur dont le chemin est "/login" est redirigée vers reCAPTCHA Enterprise au niveau de la règle Cloud Armor, ensuite reCAPTCHA Enterprise associe un cookie d'exception au navigateur de l'utilisateur qui réussit l'évaluation reCAPTCHA.
Google Cloud Armor autorise l'accès aux requêtes comportant des cookies d'exception valides.

Utiliser l'IA pour filtrer les requêtes avec adaptative protection

Adaptative protection est probablement l'une des fonctions les plus intéressantes à utiliser et à mettre en place pour protéger vos services.
Dans sa version standard, cette fonctionnalité vous enverra des alertes lorsque Google détecte une activité anormale au niveau de vos load balancer.
Dans sa version Plus (cf. Formules Cloud Armor et FinOps), la fonctionnalité va, en plus de détecter les activités anormales, être capable de vous proposer de nouvelles règles de blocage à mettre en place pour bloquer cette activité et éviter qu'elle n'atteigne vos services.
Cloud armor vous détaillera également quelle proportion du trafic anormal sera détectée par cette nouvelle règle, mais aussi une évaluation du trafic légitime bloqué par cette dernière.
Lors de la mise en place d'une de ses règles suggérées par Cloud Armor, l'utilisation de la fonctionnalité de preview décrite précédemment vous permettra d'avoir une mesure pertinente de l'impact sur votre trafic légitime ou non avant de l'activer. Vous pouvez également utiliser la fonctionnalité de limitation de nombre de requêtes afin de ne pas bloquer du trafic légitime.

Attention toutefois, par défaut, Google vous propose une règle avec un identifiant sans vous donner le détail de l'implémentation. Si vous souhaitez avoir plus d'informations et de détails sur la règle proposée, il reste possible de faire la demande auprès de Google pour activer cette fonctionnalité.

Tarification Cloud Armor et FinOps

Cloud Armor existe en deux formules qui auront deux méthodes facturations différentes:

- Dans sa version standard, Cloud Armor est facturé en mode "Pay as you go", et sera donc facturé à l'usage selon trois critères :
  - Nombre de requêtes reçues
  - Nombre de security policy déployées
  - Nombre de règles déployées
- En version Plus, Cloud Armor demande un engagement minimum de 1 an, et sera facturé avec un montant mensuel incluant la protection de 100 ressources de backend, les ressources supplémentaires étant facturées en supplément. Vous serez également facturé pour le trafic sortant de vos ressources avec des tarifs dépendant de chacune des ressources. Si le contenu diffusé est utilisé avec Cloud CDN et est considéré comme pouvant être mis en cache, les frais de traitement des données pour Cloud CDN sont appliqués pour ce contenu.

L'offre standard est particulièrement intéressante si vous souhaitez vous lancer dans Cloud Armor, sans avoir de gros besoins, ou si vous n'avez pas de visibilité à long terme sur l'utilisation de Cloud Armor.
En revanche, comme toujours sur le Cloud, surveillez bien les coûts. Si votre adoption de Cloud Armor devient plus importante et que vous multipliez les règles, la formule Plus devient plus intéressante, et vous permettra de bénéficier de l'adaptive protection.

MISE EN PLACE DE CLOUD CDN

Cloud CDN va vous permettre de décharger vos services de certaines réponses pour améliorer la performance réseau.

Attention ! L'utilisation combinée de Cloud Armor et de Cloud CDN peut avoir des effets inattendus. Cloud CDN est la première brique qui sera sollicitée pour répondre aux requêtes de vos utilisateurs. Il peut arriver de faire une erreur sur Cloud Armor qui bloquera les accès légitimes à

vos services. Dans ce cas la réponse sera mise en cache est servie par Cloud CDN.

Choisir le mode de cache

Lors de la mise en place de Cloud CDN, vous avez la possibilité de choisir parmi 3 modes de cache différents : CACHE_ALL_STATIC, USE_ORIGIN_HEADERS et FORCE_CACHE_ALL. La configuration se fait au niveau de votre service de backend

```
cdn_policy {
 cache_mode = "USE_ORIGIN_HEADERS"
 cache_key_policy {
  include_host = true
  include_protocol = true
  include_query_string = true
 }
}
```

Personnaliser les clefs de cache

Par défaut, les services de backend configurés pour utiliser Cloud CDN incluent tous les composants de l'URI de requête dans des clefs de cache pour identifier les entrées de cache. Néanmoins, vous pouvez personnaliser la clef de cache en excluant un ou plusieurs composants (le protocole, l'hôte, la chaîne de requête, en-têtes HTTP, des cookies nommés…).

Autres options de Cloud CDN

Google Cloud CDN offre la possibilité de configurer la mise en cache négative à l'aide de l'option —negative-caching. La mise en cache négative s'applique à des codes d'état spécifiques : 300, 301, 308, 302, 307, 404, 405, 410, 451 et 501.

Google Cloud CDN permet également de diffuser du contenu lorsque le serveur d'origine est inaccessible ou renvoie des erreurs à Cloud CDN à l'aide de l'option —serve-while-stale=**SECONDS**.

MONITORING

L'activation de Google Cloud Armor n'est qu'une première étape, il est important d'ajuster et d'améliorer en permanence les règles.

Pour surveiller l'état et les volumes de trafic des requêtes (autorisées, bloquées ou pré-visualisées) par stratégie ou par service de backend, Google a mis en place un tableau de bord '**Network Security Policies**' préconfiguré dans Cloud Monitoring.
Vous pouvez également consulter les logs des security policies à l'aide de cette requête :

```
resource.type:(http_load_balancer) AND jsonPayload.enforcedSecurityPolicy.
name:(SecurityPolicyName)
```

dans Cloud Logging et créer vos propre metrics et alertes à partir des logs.
Google fournit également un tableau de bord pour superviser google cloud CDN, disponible dans le repo git **monitoring-dashboard-samples**: https://github.com/GoogleCloudPlatform/monitoring-dashboard-samples/blob/master/dashboards/networking/cloud-cdn-monitoring.json

En plus de cela, vous pouvez consulter l'état de succès (hit), de défaut (miss) ou de revalidation de cache d'une requête diffusée par Cloud CDN, à l'aide de ces requêtes :

Succès de cache (hit) :

```
jsonPayload.statusDetails=("response_from_cache" OR "byte_range_caching")
```

ou

```
httpRequest.cacheHit=true
httpRequest.cacheValidatedWithOriginServer!=true
```

Succès de cache (hit) validé avec le serveur d'origine :

```
jsonPayload.statusDetails="response_from_cache_validated"
```

ou

```
httpRequest.cacheHit=true
httpRequest.cacheValidatedWithOriginServer=true
```

Défaut de cache (miss) :

```
jsonPayload.statusDetails="response_sent_by_backend"
```

ou

```
httpRequest.cacheHit!=true
httpRequest.cacheLookup=true
```

CONCLUSION

Google Cloud Armor et Google Cloud CDN sont des outils puissants permettant d'améliorer considérablement la performance et la sécurité de votre site ou application web. Cloud armor offre une solution de sécurité robuste et personnalisable qui aide à se protéger contre les attaques tout en permettant une surveillance en temps réel pour détecter les menaces potentielles. Cloud CDN, quant à lui, assure une diffusion globale efficace du contenu et des temps de chargement plus rapides. Il est fortement recommandé de les considérer pour tout site ou application web déployés sur Google Cloud.



1 an de Programmez!

ABONNEMENT PDF : 45 €

Abonnez-vous directement sur
www.programmez.com

Estimez vos coûts Azure avec Infracost et Terraform

L'apparition de l'Infrastructure as Code permet de montrer qu'il est possible de provisionner et de déployer une infrastructure cloud à très grande échelle et très rapidement. Avec cette émergence rapide des infrastructures, les entreprises rencontrent une problématique qui est l'estimation des coûts de leur infrastructure qui va être déployée.



Mikael Krief
Consultant Devops freelance / Microsoft MVP Azure / HashiCorp Ambassador

Parmi tous les outils et langages d'infrastructure as Code (IaC), Terraform de HashiCorp devient très populaire par son utilisation et sa grande communauté. Un des avantages de Terraform est de pouvoir avoir une prévisualisation des changements apportés sur l'infrastructure avant même de les appliquer.

Il est possible de pouvoir estimer les coûts de ressources cloud en même temps que son développement (appelé aussi configuration) avec Terraform avant d'appliquer ses changements ?

Pour répondre à cette question, nous allons présenter un outil appelé Infracost qui a pour but d'avoir une estimation des coûts de ressources qui sont codés dans une configuration Terraform.

Dans cet article nous allons voir comment installer Infracost, son utilisation en ligne de commande, et comment utiliser son extension pour VSCode.

Présentation de Infracost

Infracost (https://www.infracost.io/) se compose de 3 composants :

- Le binaire qui est open source et gratuit et qui permet toutes les opérations de bases d'estimation de prix des ressources.
- Infracost Cloud qui est une solution SaaS qui permet en plus de la version gratuite d'avoir des fonctionnalités de CI/CD intégré, sur les Pull Requests, d'avoir des policies, une intégration avec Jira et bien d'autres fonctionnalités pour une meilleure intégration en équipe. Les informations sur le prix de cette version sont disponibles ici https://www.infracost.io/pricing/
- L'API de Infracost qui se connecte aux différents clouds pour récupérer les informations de prix des différentes ressources cloud.

Les clouds pris en charge sont Azure, AWS et GCP. Dans la suite de cet article, nous verrons un cas d'usage avec Azure. De plus, Infracost permet aussi une intégration dans plusieurs systèmes de CI/CD comme GitHub Action, GitLab, Azure Pipelines, Jenkins, CircleCI et Atlantis.

Installation d'Infracost

Après cette brève introduction, nous allons installer le binaire. Sa procédure d'installation dépend du système sur lequel vous travaillez localement, mais aussi du mode d'installation : manuel ou automatisé via un script. Toutes les procédures d'installation de Infracost sont disponibles sur cette documentation : https://www.infracost.io/docs/. Nous allons l'installer sur un Linux avec ce script :

```
curl -fsSL https://raw.githubusercontent.com/infracost/infracost/master/scripts/install.sh | sh
```

Figure 1

Maintenant que Infracost est installé, nous allons générer une clé d'API.

Authentification et génération de l'API Key

Pour fonctionner, Infracost a besoin d'être authentifié et de générer une clé d'API, qui servira à récupérer les informations de prix des providers cloud.

Pour ceci, nous exécutons la commande suivante :

```
infracost auth login
```

La sortie de cette commande ouvre une page web (ou nous donne une URL) dans un navigateur web : **Figure 2**

Puis, on clique sur le bouton « Log in » pour s'authentifier de façon anonyme : **Figure 3**

Une confirmation s'affiche dans la page du navigateur ainsi que dans le Terminal de commande.

Dans le terminal de commande nous avons aussi l'information que l'API KEY est générée ainsi que son fichier : **Figure 4**

Nous pouvons maintenant exécuter Infracost sur une configuration Terraform.

```
mikael@vmdev-linux:~$ curl -fsSL https://raw.githubusercontent.com/infracost/infracost/master/scripts/install.sh | sh
Downloading latest release of infracost-linux-amd64...

Validating checksum for infracost-linux-amd64...

Moving /tmp/infracost-linux-amd64 to /usr/local/bin/infracost (you might be asked for your password due to sudo)

Completed installing Infracost v0.10.15
```

Figure 1

```
mikael@vmdev-linux:~$ infracost auth login
We're redirecting you to our log in page, please complete that,
and return here to continue using Infracost.

If the redirect doesn't work, use this URL:

https://dashboard.infracost.io/login?cli_port=39111&cli_state=be2dd34c-c4d6-4b43-910f-10de67921e71&cli_version=v0.10.15&os=linux&utm_source=cli

Waiting...
```

Figure 2

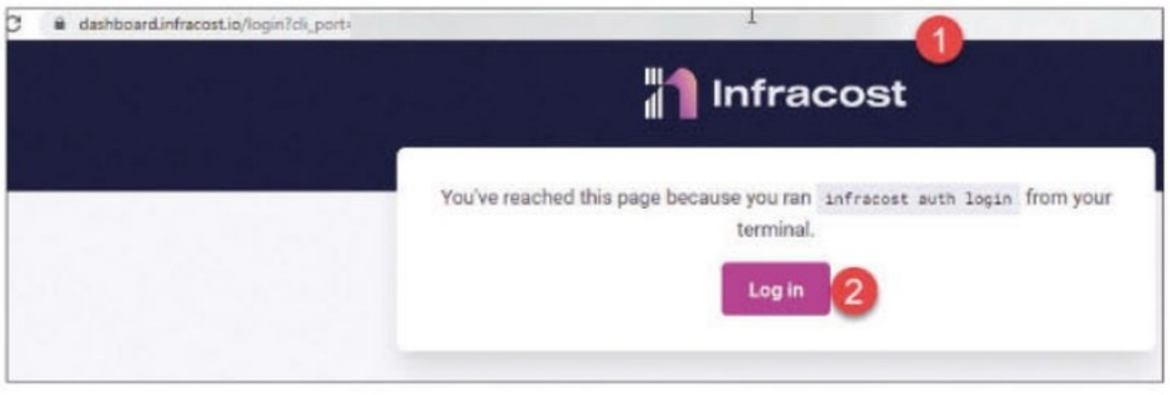


Figure 3

Première estimation des coûts d'une infrastructure Azure

Nous allons maintenant faire une estimation de coûts d'une infrastructure Azure qui est composée des ressources suivantes :

- 1 Resource Groupe
- 1 Virtual Network, d'un Subnet
- 1 Virtual Machine Linux de type Standard_DS2_V2.
- 1 Service plan de type Standard S1
- 1 web App Linux

Toute cette infrastructure est codée dans une configuration Terraform dont le code source de cette configuration

Figure 4

```
The API key was saved to /home/mikael/.config/infracost/credentials.yml

Your account has been authenticated. Run Infracost on your Terraform project by running:
```

```
mikael@DESKTOP-MIKAEL:.../src$ infracost breakdown --path .
Evaluating Terraform directory at .
  ✔ Downloading Terraform modules
  ✔ Evaluating Terraform directory
  ✔ Retrieving cloud prices to calculate costs

Project: .

 Name                                                Monthly Qty  Unit                     Monthly Cost

 azurerm_app_service_plan.plan-app
 └─ Instance usage (S1)                                      730  hours                          $73.00

 azurerm_linux_virtual_machine.vm
 ├─ Instance usage (pay as you go, Standard_Ds2_V2)          730  hours                          $99.28
 └─ os_disk
    ├─ Storage (S4)                                            1  months                         $1.54
    └─ Disk operations                          Monthly cost depends on usage: $0.0005 per 10k operations

 OVERALL TOTAL                                                                                 $173.82

7 cloud resources were detected:
∙ 2 were estimated, 1 of which usage-based costs, see https://infracost.io/usage-file
∙ 4 were free, rerun with --show-skipped to see details
∙ 1 is not supported yet, rerun with --show-skipped to see details
```

Figure 5

```
mikael@DESKTOP-MIKAEL:.../src$ infracost breakdown --path . --show-skipped
Evaluating Terraform directory at .
  ✔ Downloading Terraform modules
  ✔ Evaluating Terraform directory
  ✔ Retrieving cloud prices to calculate costs

Project: .

 Name                                                Monthly Qty  Unit                     Monthly Cost

 azurerm_app_service_plan.plan-app
 └─ Instance usage (S1)                                      730  hours                          $73.00

 azurerm_linux_virtual_machine.vm
 ├─ Instance usage (pay as you go, Standard_Ds2_V2)          730  hours                          $99.28
 └─ os_disk
    ├─ Storage (S4)                                            1  months                         $1.54
    └─ Disk operations                          Monthly cost depends on usage: $0.0005 per 10k operations

 OVERALL TOTAL                                                                                 $173.82

7 cloud resources were detected:
∙ 2 were estimated, 1 of which usage-based costs, see https://infracost.io/usage-file
∙ 4 were free:
  ∙ 1 x azurerm_network_interface
  ∙ 1 x azurerm_resource_group
  ∙ 1 x azurerm_subnet
  ∙ 1 x azurerm_virtual_network
∙ 1 is not supported yet, see https://infracost.io/requested-resources:
  ∙ 1 x azurerm_linux_web_app
```

Figure 6

```
resource "azurerm_linux_virtual_machine" "vm" {
  name                            = "vm-demo"
  resource_group_name             = azurerm_resource_group.rg.name
  location                        = azurerm_resource_group.rg.location
  size                            = "Standard_E8_v4"
  disable_password_authentication = false
}
```

Figure 7

Terraform est disponible ici https://gist.github.com/mikaelkrief/634104974f45c34eec3686124d3ce851

Pour obtenir notre estimation de coûts avant d'exécuter Terraform et créer ces ressources, nous allons dans le dossier qui contient cette configuration Terraform, exécuter Infracost avec la commande suivante :

```
infracost breakdown —path .
```

Cette commande affiche en sortie une estimation de coûts des différentes ressources Azure : **Figure 5**

D'après ces estimations, nous payerons 173,82 $ / mois, pour le service plan ainsi que la Virtual Machine et son disque OS.

À la fin de cette exécution, Infracost indique qu'il a pu faire l'estimation pour 2 ressources, que 4 ressources sont gratuites et que 1 ressource n'est pas encore supportée dans l'analyse.

Pour avoir plus de détails sur les autres ressources non estimées, on exécute la commande :

```
infracost breakdown —path . —show-skipped
```

On obtient le résultat suivant : **Figure 6**

Infracost indique en détail les 4 ressources gratuites ainsi que la ressource non supportée.

L'estimation de coûts nous convient, nous pouvons provisionner cette infrastructure en exécutant le workflow de Terraform en exécutant les commandes suivantes :

```
terraform init, plan et apply
```

L'infrastructure est provisionnée, mais elle va évaluer au cours du temps, l'objectif maintenant est d'estimer les coûts uniquement des changements apportés sur cette infrastructure.

Pour cela nous allons sauvegarder l'état de notre estimation dans un fichier au format Json en exécutant la commande suivante :

```
infracost breakdown —path . —format json —out-file infracost-base.json
```

Cette commande fait exactement la même opération d'estimation que précédemment, mais à sa différence exporte le détail de l'estimation dans un fichier Json au lieu de l'afficher dans le terminal.

Faisons évoluer la configuration de notre infrastructure et regardons les changements de coûts.

Estimation des changements de l'infrastructure

Nous allons modifier la configuration Terraform de notre infrastructure pour choisir une taille de Virtual machine supérieure pour augmenter les performances en CPU et RAM.

Nous modifions le code Terraform pour utiliser une VM de taille Standard_E8_v4 (au lieu de Standard_DS2_v2) comme le montre l'extrait de configuration Terraform ci-dessous : **Figure 7**

L'objectif est maintenant d'avoir une estimation des coûts supplémentaire (ou en moins) qui me seront appliqués lorsque nous aurons exécuté ce changement de code.

Pour cela, dans le terminal on exécute la commande suivante :

```
infracost diff —path . —compare-to infracost-base.json
```

Cette commande compare les coûts de l'infrastructure de cette configuration Terraform avec les coûts précédents qui ont été exportés dans le fichier Json. L'image suivante montre le résultat de son exécution : **Figure 8**

Nous voyons bien que nous allons passer de 101$ à 444$ par mois. À l'inverse si nous choisissons une taille de VM plus petite soit du Standard_B2s, cette commande renvoie l'estimation suivante : **Figure 9**

L'estimation rapporte que nous paierons 64$/mois en moins.

Nous venons de voir comment utiliser les lignes de commande pour avoir une estimation des coûts d'une infrastructure configurée avec Terraform, regardons son intégration dans VSCode.

Intégration dans VSCode

À part son binaire CLI, Infracost possède aussi une extension à VSCode (https://code.visualstudio.com/) qui permet d'avoir une estimation des coûts des ressources Azure configurés avec Terraform directement dans VSCode et ainsi d'avoir une estimation en même temps que nous écrivons le code Terraform.

La documentation de cette extension est disponible ici https://marketplace.visualstudio.com/items?itemName=Infracost.infracost et elle peut être installée directement à partir de VSCode, dans la gestion des extensions : **Figure 10**

Une fois l'extension installée, nous cliquons sur l'icône de Infracost dans le menu de gauche et nous procédons à l'authentification de Infracost : **Figure 11**

Une fois connecté, nous ouvrons notre code de configuration Terraform avec VSCode. Nous observons que l'estimation du coût s'affiche automatiquement sur la ressource Azure Terraform concernée : **Figure 12**

Si on clique sur cette ligne, une fenêtre s'ouvre dans VSCode avec les détails du coût pour cette ressource : **Figure 13**

Puis, en modifiant la taille de la VM (propriété size), toute l'estimation de coût se met à jour.

Conclusion

Nous avons vu dans cet article une utilisation simple de Infracost afin d'estimer les coûts d'une infrastructure Azure qui va être provisionnée avec Terraform. Infracost possède bien plus de fonctionnalités comme l'intégration dans une CI/CD, ainsi que sa solution cloud Infracost Cloud.

Je vous invite à lire sa documentation https://www.infracost.io/docs/ pour en savoir plus.

```
mikael@DESKTOP-MIKAEL:.../src$ infracost diff --path . --compare-to infracost-base.json
Evaluating Terraform directory at .
  ✔ Downloading Terraform modules
  ✔ Evaluating Terraform directory
  ✔ Retrieving cloud prices to calculate costs

Project: .

~ azurerm_linux_virtual_machine.vm
  +$345 ($101 → $445)

    ~ Instance usage (pay as you go, Standard_Ds2_V2 → Standard_E8_v4)
      +$345 ($99.28 → $444)

Monthly cost change for .
Amount:  +$345 ($101 → $445)
Percent: +342%
```

Figure 8

```
mikael@DESKTOP-MIKAEL:.../src$ infracost diff --path . --compare-to infracost-base.json
Evaluating Terraform directory at .
  ✔ Downloading Terraform modules
  ✔ Evaluating Terraform directory
  ✔ Retrieving cloud prices to calculate costs

Project: .

~ azurerm_linux_virtual_machine.vm
  -$64.24 ($101 → $36.58)

    ~ Instance usage (pay as you go, Standard_Ds2_V2 → Standard_B2s)
      -$64.24 ($99.28 → $35.04)

Monthly cost change for .
Amount:  -$64.24 ($101 → $36.58)
Percent: -64%
```

Figure 9

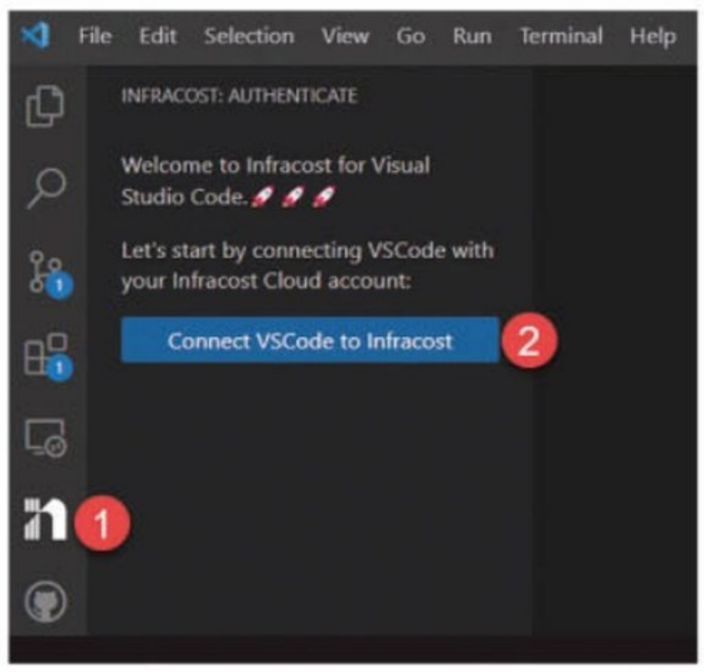


Figure 11

Figure 10

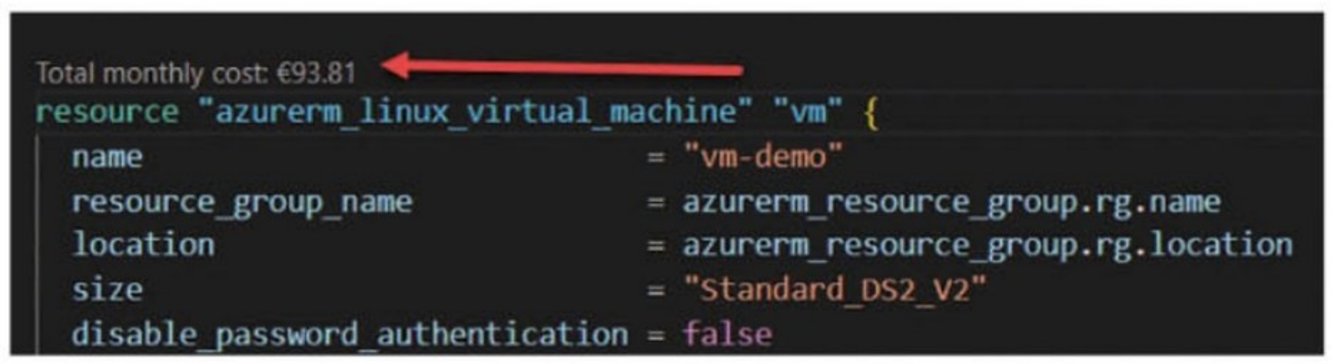


Figure 12

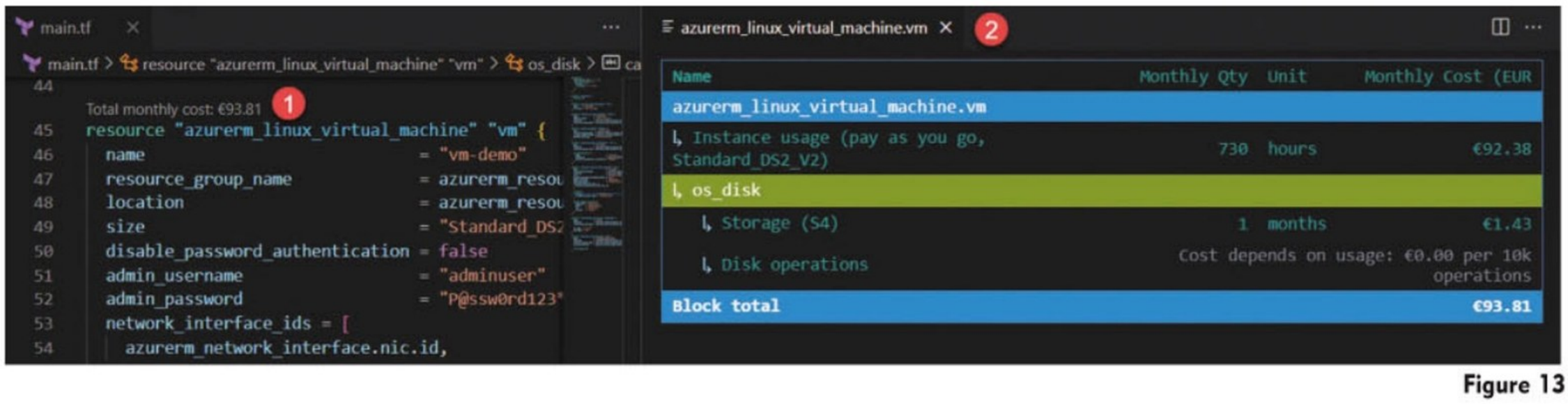
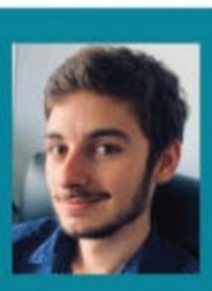


Figure 13

Propulsez votre app Angular dans le cloud avec Code Space



Pierre Bouillon

Ingénieur logiciel Full Stack développant principalement des applications web avec .NET et Angular.

https://pbouillon.github.io

Développant divers projets sur différents ordinateurs, j'ai pour habitude de passer d'environnement en environnement. En conséquence, je dois fréquemment mettre à jour mes dépendances et branches locales pour éviter le moindre conflit dans mes sources ou la moindre perte de mon travail. Dernièrement, au cours du développement de l'un de ces projets, j'ai découvert une manière d'abstraire ces contraintes en tirant parti de Docker, VS Code et GitHub. Voyons ensemble comment !

Créer notre application Angular

La première étape n'a rien d'extravagant et nous allons donc simplement nous contenter de créer un nouveau projet vide pour notre exemple :

```
~$ ng new code-space-demo
```

Ne vous préoccupez pas des options et laissez tout par défaut, le code en soit ne va pas du tout entrer en compte pour notre démo. Ouvrez ensuite votre projet dans VS Code pour y voir alors l'arborescence : **Figure 1**

Créer notre *Dev Container*

Notre Dev-quoi ?

Les Dev Containers (https://containers.dev/) sont une (plus si) nouvelle fonctionnalité de VS Code permettant à votre éditeur de tourner dans un environnement dockerisé.

Simplement en configurant une image Docker pour votre projet en plus d'éventuels paramètres additionnels, vous pouvez lancer VS Code dans une instance entièrement configurée avec l'entièreté de vos dépendances et paramétrages préappliqués pour votre projet.

Configurer VS Code

Comme pour n'importe quelle autre manière d'étendre le fonctionnement de VS Code, cette fonctionnalité est disponible au travers d'une extension.

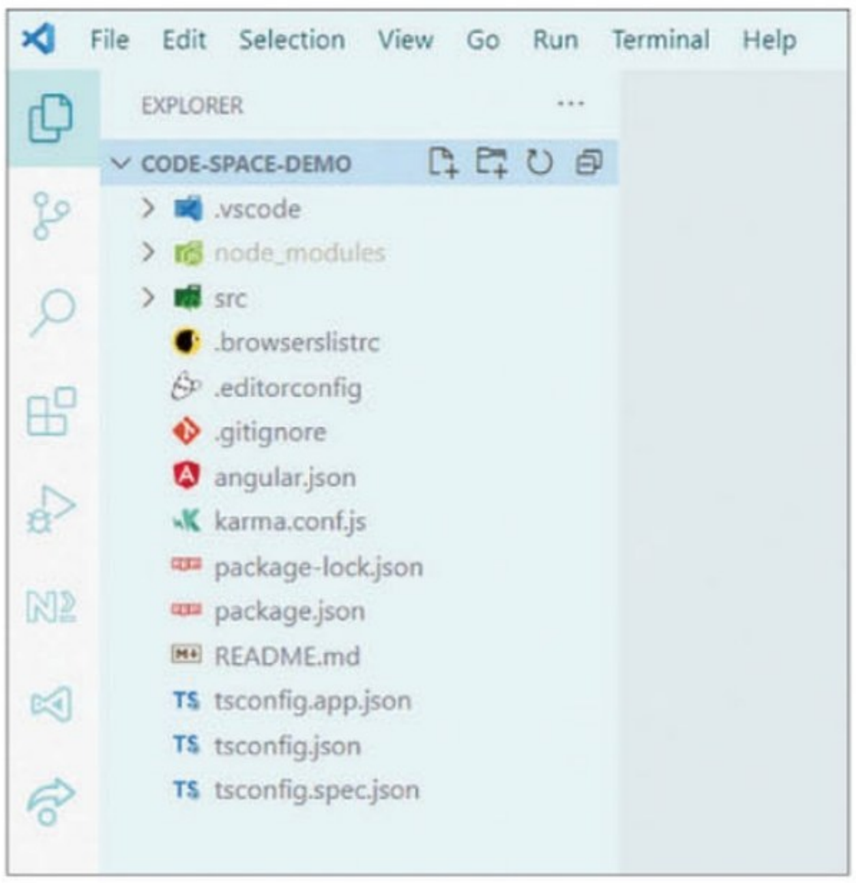


Figure 1

Ouvrez le panneau de navigation des extensions et cherchez « Dev Containers » : **Figure 2**

Nous voilà prêts pour la suite !

Ajouter un Dev Container à notre application

Depuis la palette de commande (ctrl + shift + P), cherchez « Dev Containers : Add Dev Container Configuration Files » :

Puis, dans la fenêtre de dialogue suivante, choisissez la configuration propre aux développements avec NodeJS et TypeScript : **Figure 3**

Enfin, sélectionnez la dernière version de Node et passez les derniers imports optionnels proposés. **Figure 4**

Il est à noter que, depuis les dernières versions de Dev Containers, il est maintenant possible d'importer certaines configurations unitaires de votre conteneur appelées *features* (https://containers.dev/implementors/features/).

Pour Angular par exemple, la *feature* « Angular CLI » est disponible à la sélection. Cependant, pour ce tutoriel, nous ne l'utiliserons pas. En revanche, n'hésitez pas à explorer les *features* disponibles pour vos projets à l'avenir ! **Figure 5**

Une fois ces étapes terminées, de nouveaux fichiers devraient apparaître sous un nouveau dossier .devcontainer : **Figure 6**

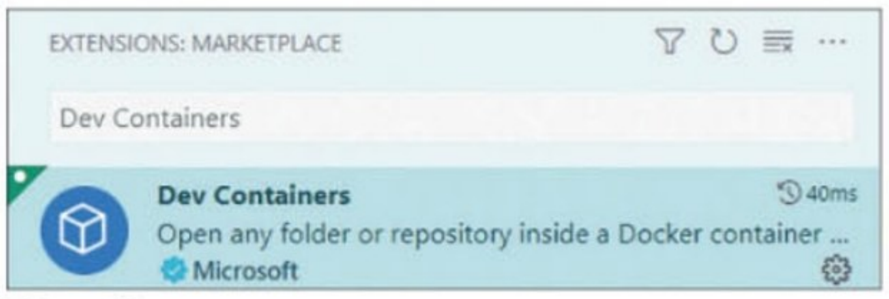


Figure 2



Figure 3

VS Code devrait également vous indiquer qu'il a détecté votre configuration pour un Dev Container et vous propose alors de redémarrer en l'utilisant : **Figure 7**

Si l'option n'apparaît pas, vous pouvez vous-même achever le même résultat via la palette de commande en cherchant « Dev Containers: Rebuild and Reopen in Container »

Assurez-vous que Docker est bien lancé sur votre machine et rouvrez alors votre éditeur dans le conteneur, comme proposé par VS Code. Une fois redémarré, votre interface devrait être sensiblement similaire à celle-ci : **Figure 8**

Est-ce que l'on peut coder maintenant ?

Commençons par lancer notre application depuis le terminal de VS Code, qui tourne dans son conteneur : **Figure 9**

Bien que ce soit décevant, il n'y a ici pas vraiment de surprise : Angular ne peut pas être lancé puisque notre conteneur n'a pas la moindre notion de ce qu'est Angular à ce moment, et donc encore moins de son CLI. Heureusement, il va être assez facile d'y remédier en configurant notre Dev Container. Depuis le Dockerfile, il nous est possible d'ajouter des dépendances là où les commentaires nous indiquent que cela est possible :

```
# [Optional] Uncomment if you want to install more global node packages
# RUN su node -c "npm install -g <your-package-list-here>"
RUN su node -c "npm install -g @angular/cli@14.1.3"
```

Comme habituellement pour la création de vos images, précisez ici le numéro de version exact des dépendances que vous allez utiliser peut grandement vous aider à avoir un environnement prédictible et cohérent sur l'ensemble de vos machine.

Une fois vos dépendances installées, sélectionnez alors « Rebuild container » depuis la palette de commande et essayez à nouveau de lancer votre application : **Figure 10**

Et voici notre environnement Angular configuré !

En réalité il est *presque* configuré. Si vous utilisez Cypress, vous allez devoir également paramétrer l'installation de ses dépendances dans le Dockerfile.

Après un rapide coup d'œil à celles requises, vous pourriez le faire en ajoutant le code suivant :

```
# Cypress dependencies
RUN apt-get update && \
  apt-get -y install —no-install-recommends \
  libgtk2.0-0 \
  libgtk-3-0 \
  libgbm-dev \
  libnotify-dev \
  libgconf-2-4 \
  libnss3 \
  libxss1 \
  libasound2 \
  libxtst6 \
  xauth \
  xvfb

# Cypress
RUN su node -c "npm install -g cypress"
```

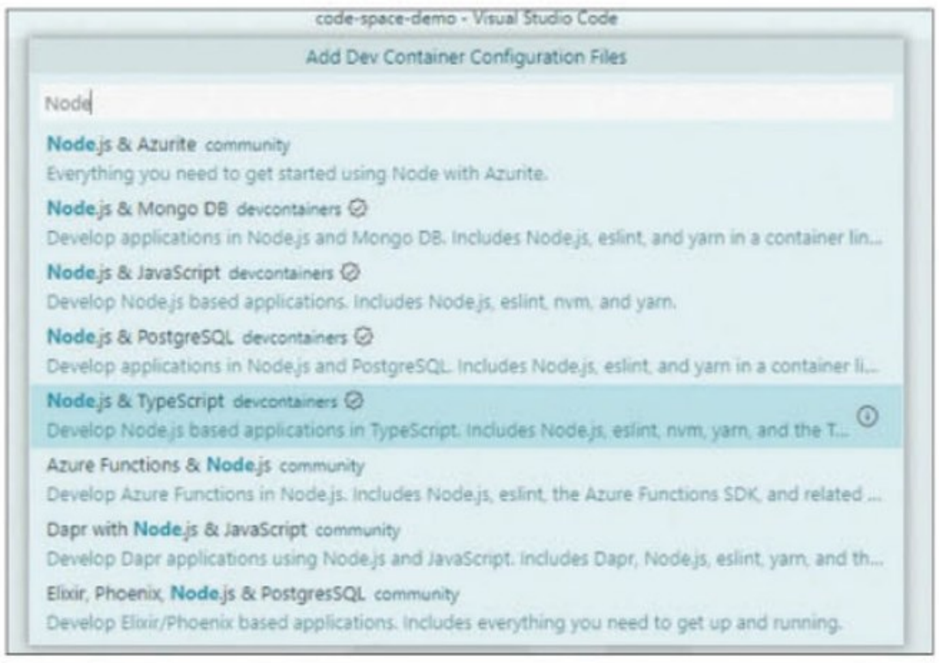


Figure 4

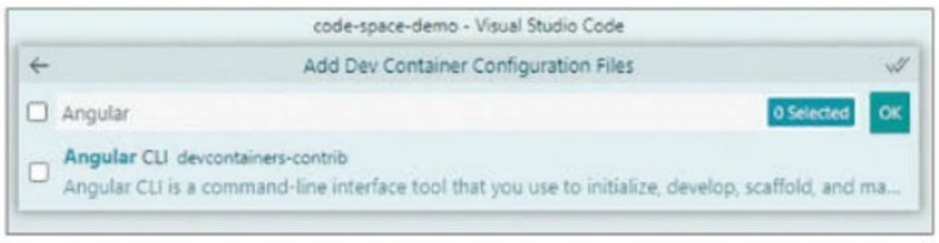


Figure 5

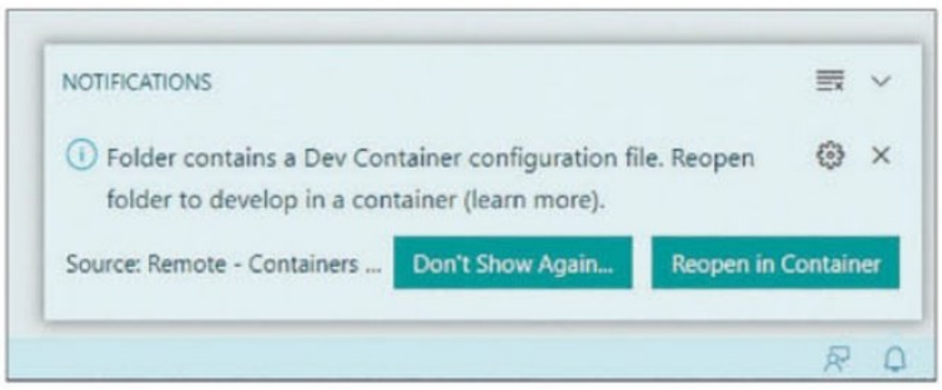


Figure 7



Figure 9

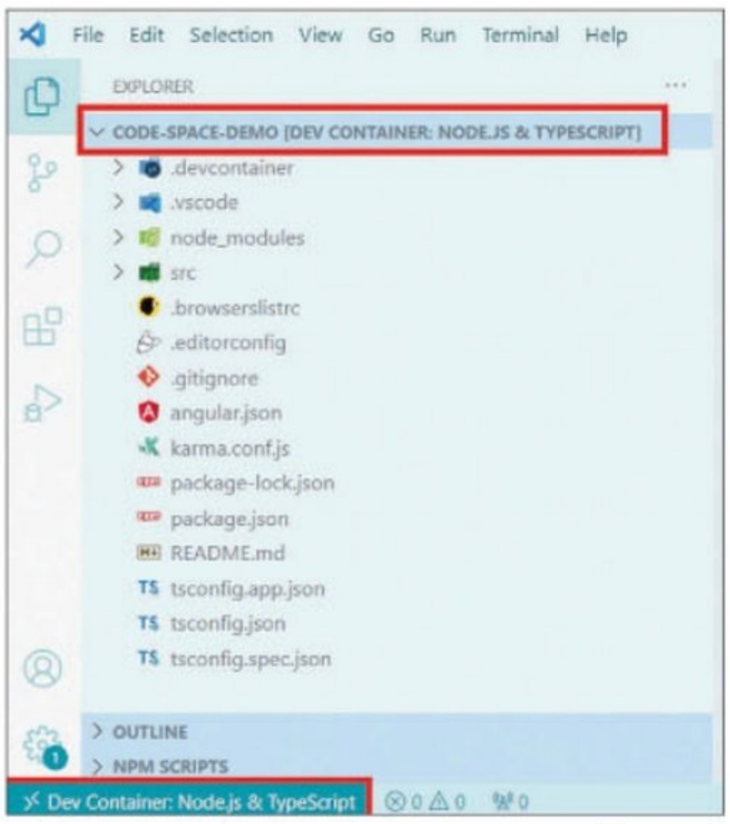


Figure 8

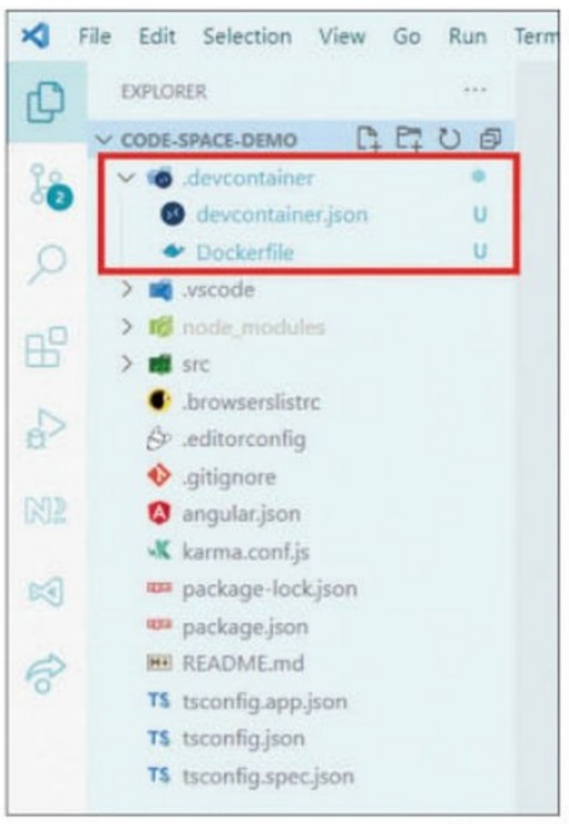


Figure 6

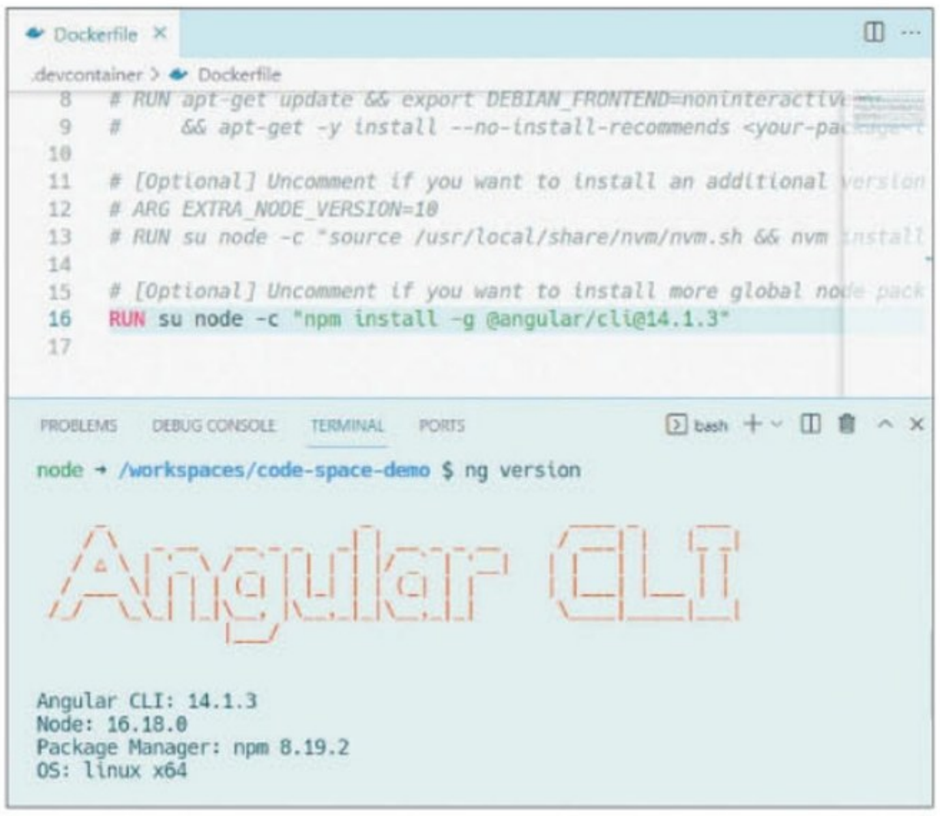


Figure 10

Configurer notre IDE

Maintenant que nos dépendances techniques sont correctement spécifiées, il pourrait être intéressant de configurer l'éditeur lui-même pour accompagner notre développement futur.

Vous avez peut-être noté qu'un second fichier se trouve dans le dossier .devcontainer : le fichier devcontainer.json. Dans ce fichier, il nous est possible de configurer le comportement de l'instance de VS Code qui tournera dans notre Dev Container.

Par exemple, faire tourner notre application Angular dans notre conteneur est déjà bien, mais pouvoir y accéder localement serait encore mieux. Pour ce faire, il nous suffit de simplement préciser les ports auxquels nous souhaitons accéder localement :

```
"forwardPorts": [ 4200 ]
```

Puisqu'il s'agit ici d'un tableau, vous pouvez bien entendu exposer plusieurs ports à la fois.

C'est déjà mieux ! En revanche, lorsque nous créons notre conteneur, nous devons installer l'ensemble des dépendances. Il pourrait être intéressant que VS Code puisse le faire lui-même à la suite de la création de son instance, et c'est exactement ce que l'option postCreateCommand permet de faire :

```
"postCreateCommand": "npm install"
```

Malheureusement ce n'est cette fois pas un tableau et une seule commande peut être lancée ici. Si vous souhaitez faire plusieurs appels ou exécuter une logique plus complexe, une solution peut-être de placer ces derniers dans un script à part et de l'appeler à cet endroit.

Enfin, il nous est également possible de préciser les extensions qui seront installées par défaut dans notre instance en complétant le tableau extensions, de manière à avoir un environnement de développement prêt et configuré spécialement pour notre projet dès son lancement.

Au final, voici à quoi devrait ressembler votre devcontainer.json à l'issue de ces étapes :

```
{
 "name": "Node.js & TypeScript",
 "build": {
  "dockerfile": "Dockerfile",
  "args": {
   "VARIANT": "16-bullseye"
  }
 },
 "settings": {},
 "extensions": [
  "angular.ng-template",
  "dbaeumer.vscode-eslint",
  "esbenp.prettier-vscode",
  "shardulm94.trailing-spaces",
  "firefox-devtools.vscode-firefox-debug"
 ],
 "forwardPorts": [ 4200 ],
 "postCreateCommand": "npm install",
 "remoteUser": "node"
}
```

Lançons alors la régénération de notre conteneur et observons si des changements apparaissent :

- Au démarrage, le terminal s'ouvre et affiche le résultat du lancement de notre commande npm install : **Figure 11**
- Dans la section PORTS, nous pouvons également observer que ceux spécifiés sont bien redirigés : **Figure 12**

Il semblerait que tout est bon pour notre environnement !

Pour la suite, il vous faudra avoir un compte GitHub.

Depuis VS Code, ou n'importe quel outil de votre choix, poussez votre code sur un nouveau répertoire GitHub.

Depuis la page d'accueil de ce dernier, sous la section « Code », cliquer sur l'onglet « Codespaces » : **Figure 13**

Présentation de Codespaces

Codespaces (https://github.com/features/codespaces) est une fonctionnalité introduite par GitHub qui vous permet de travailler sur vos projets au sein d'une instance VS Code entièrement configurée et paramétrée, directement dans le cloud.

Figure 11

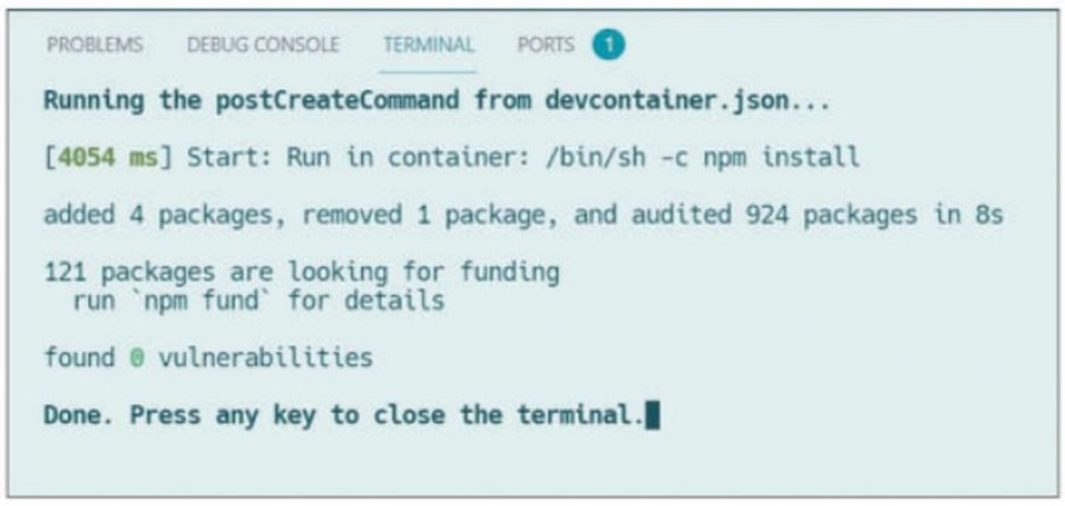


Figure 13

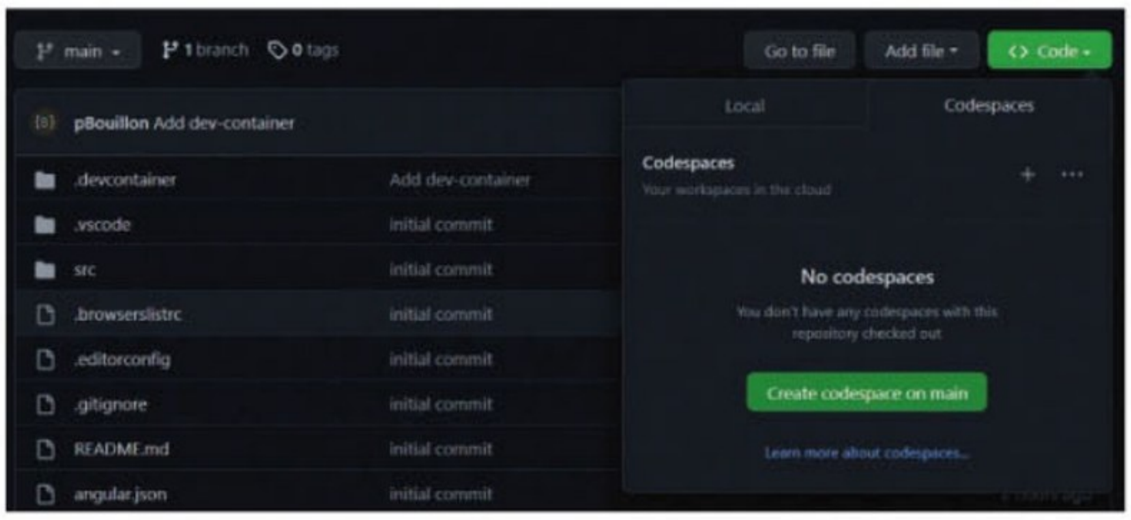




Figure 12

Il semblerait que les performances et l'efficacité semblent même suffisamment satisfaisantes pour que GitHub l'utilise pour ses propres projets (https://github.blog/2021-08-11-githubs-engineering-team-moved-codespaces).
Sur base de notre configuration dans .devcontainer, GitHub est en mesure de détecter comment notre projet est construit et peut alors démarrer une instance de VS Code directement depuis *codespaces*.
Attention cependant, notre exemple ici est *très* minimaliste et bien plus de choses peuvent être paramétrées. Vous pourriez par exemple configurer les capacités de la machine qui hébergera votre instance de VS Code, une autre image Docker spécifique, etc.
Cliquer sur « Create a codespace on main » vous ouvrira alors un nouvel onglet avec l'état de la construction de votre codespace : **Figure 14**
Quelques secondes plus tard, votre IDE est prêt et fonctionnel dans le cloud. Vous pouvez également noter qu'il vit à une URL générée par GitHub et que VS Code est conscient qu'il est dans un environnement spécifique à codespace : **Figure 15**

Une fonctionnalité "cachée" au premier coup d'œil est le support natif des redirections locales : une fois que vous aurez lancé votre application via ng serve, la CLI d'Angular vous invite à naviguer sur http://localhost:4200 (par défaut). Cependant, consulter localhost lorsque votre application tourne dans le cloud ne vous affichera aucun résultat.
Heureusement, l'instance codespace de VS Code interceptera cette interaction et vous redirigera vers une URL dédiée qui reflétera les résultats de votre commande. En cliquant sur cette URL (ctrl + clic) depuis le terminal de VS Code, vous serez alors redirigé vers un nouvel onglet pour y voir votre application : **Figure 16**

Tout semble prêt, il ne vous reste plus qu'à construire votre produit, mais à partir de maintenant c'est une tâche que je vous délègue.

Avantages et utilisations potentielles

L'utilisation de codespaces ou même seulement des Dev Containers peut réellement simplifier votre processus de développement.
Les avantages induit par l'utilisation des Dev Containers au sujet des dépendances et de la configuration d'un environnement local sont sensiblement les mêmes que les raisons pour lesquels vous pourriez vouloir utiliser Docker. Cependant, ils permettent également de configurer l'éditeur en tant que tel, vous permettant de le normaliser et de le préconfigurer bien plus rapidement entre vos nouveaux environnements de développement.
Codespaces, quant à lui, vous permet de transposer ces avantages à un niveau supérieur en faisant abstraction de l'environnement lui-même. Grâce à cette fonctionnalité, vous pouvez virtuellement coder depuis n'importe quel endroit et n'importe quel environnement local (étant donné une connexion internet suffisante, bien entendu).
Combinées, ces deux technologies vous permettent d'entièrement standardiser certains processus comme l'onboarding d'un nouveau collaborateur ou même propager rapidement et à travers de nombreux environnement de nouvelles configurations.
Un très bon cas d'usage à mon sens serait par exemple pour les entretiens techniques : migrer vos tests techniques dans un tel environnement vous permettrait de pouvoir évaluer l'ensemble de vos candidats dans des situations strictement similaires sans nécessiter de phases d'installation parfois laborieuses ou pleines de surprises, vous permettant ainsi de gagner en temps et d'éviter tout stress superflu.

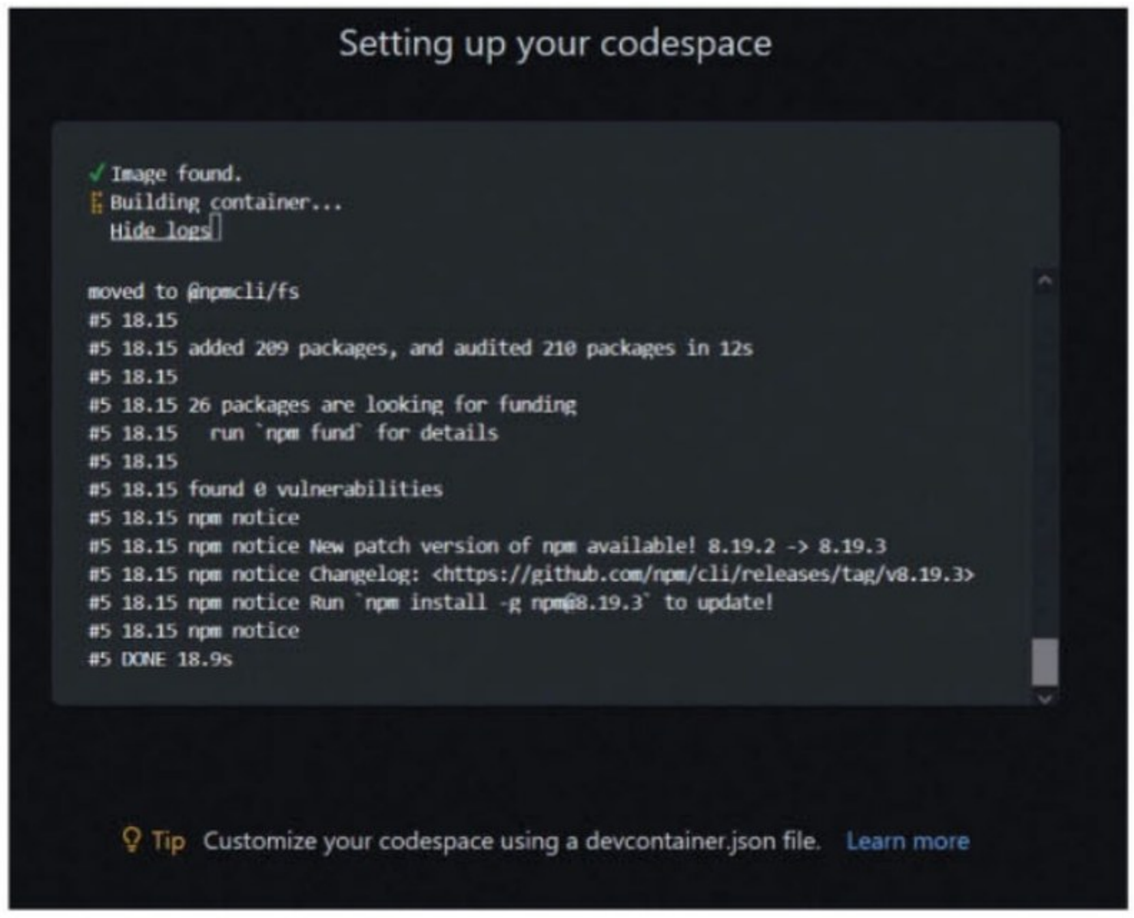


Figure 14

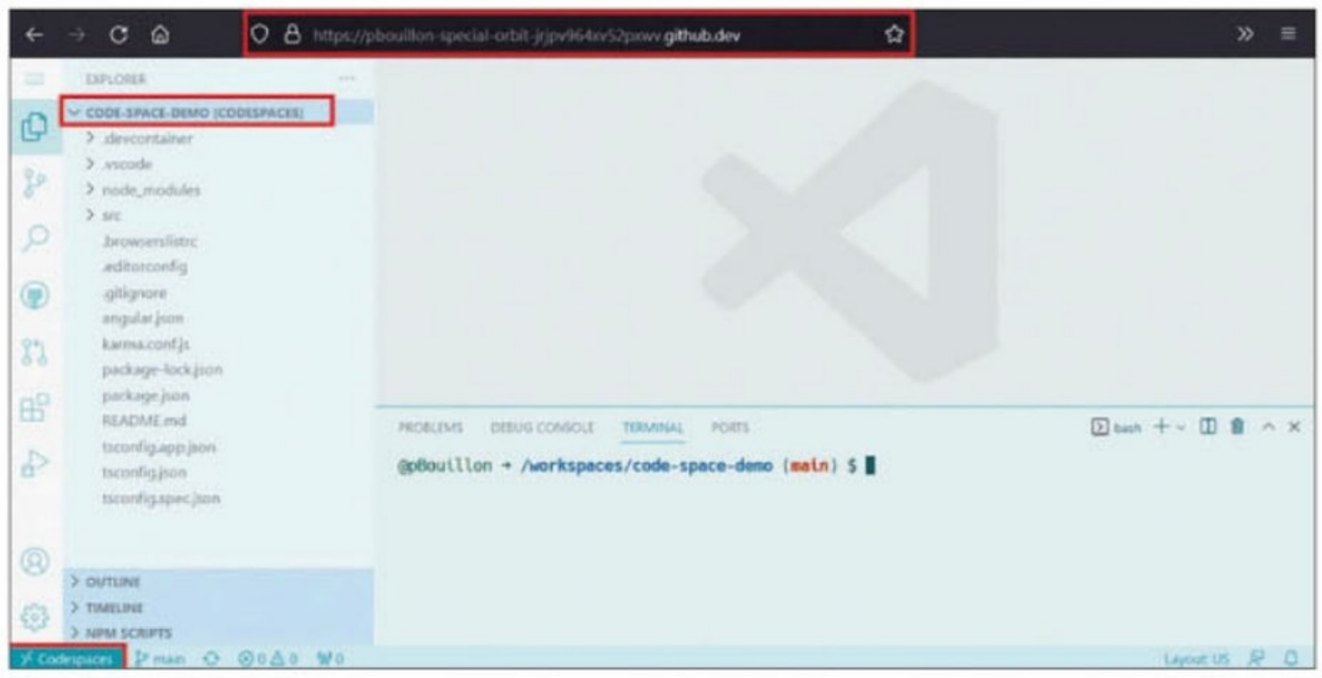


Figure 15

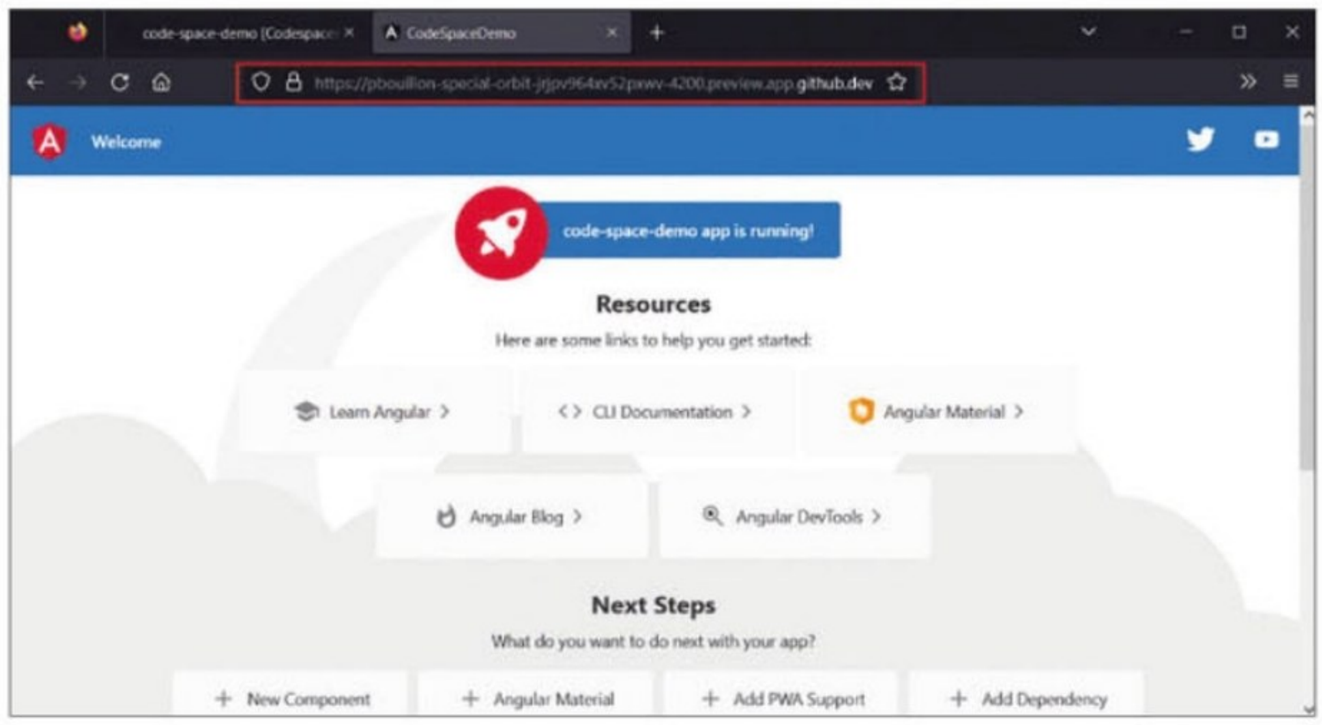


Figure 16

Azure Spring Apps Entreprise



Steve P.

Specialist Application Innovation – Microsoft

Specialist Application Innovation au sein de l'équipe MS Global Black Belt, Steve a démarré sa carrière en 2000 en tant que Développeur Java, évolua ensuite en tant qu'Architecte Java chez IBM et Sun puis Architecte Solution au sein de DSI de grands comptes avant de rejoindre Microsoft. Il est aussi Azure Spring Apps Subject Matter Expert & Java Champion.

Microsoft



Fabien MARTIN

Solution Engineer – VMware

Actuellement Solution Engineer chez VMware Tanzu, Fabien a eu un début de carrière assez classique dans l'informatique ! Développeur, Leader Technique, Architecte, il a eu l'occasion de travailler sur de nombreux projets autour du Web, du DevSecOps et du Cloud.

vmware®

En 2019 Microsoft et Pivotal (racheté par VMware) ont conjointement lancé l'offre Azure Spring Cloud afin de répondre aux défis des développeurs, des opérateurs, et des DevOps pour développer, déployer et opérer des applications Spring. En juin 2022, Microsoft et VMware décident de renommer « Azure Spring Cloud » en « Azure Spring Apps » (ou ASA) qui depuis propose un tiers Entreprise. Azure Spring Apps Entreprise combine l'expertise de la plateforme Cloud de Microsoft, avec le savoir-faire et l'innovation du portfolio VMware Tanzu.

Spring a été créé par Pivotal et est maintenant développé par VMware. Il est inclus dans le portfolio VMware Tanzu. Ce dernier offre un ensemble d'outils permettant d'adresser les problématiques que l'on rencontre dans l'univers du Cloud.

Spring est apparu en 2004 en tant que framework de développement Java, mais c'est aujourd'hui tout un écosystème de composants open source qui couvre de nombreux besoins et middlewares. Que ce soit pour la construction d'applications Web, de microservices, de développement serverless, de déploiement et d'intégration dans le cloud, il existe plus de soixante composants permettant de répondre aux différentes problématiques auxquels sont confrontés les développeurs.

Nous n'allons pas présenter l'intégralité de l'écosystème aujourd'hui, retenez simplement qu'il y a de fortes chances qu'il existe une librairie qui réponde à la problématique que vous cherchez à résoudre ! **Figure 1**

Azure Spring Apps

ASA est un service clé en main qui permet aux développeurs de déployer des applications Spring dans un environnement PRODUCTION-READY en 30 minutes avec un SLA de 99,9%.

Application et le déploiement sont les deux concepts clés du modèle de ressources d'Azure Spring Apps. Une application est l'abstraction d'une application métier. Une version de code ou de binaire déployée comme application s'exécute dans un déploiement. Plusieurs instances de l'application s'exécutent au sein d'un déploiement afin de répondre aux besoins de scalabilité et de haute disponibilité. **Figure 2**

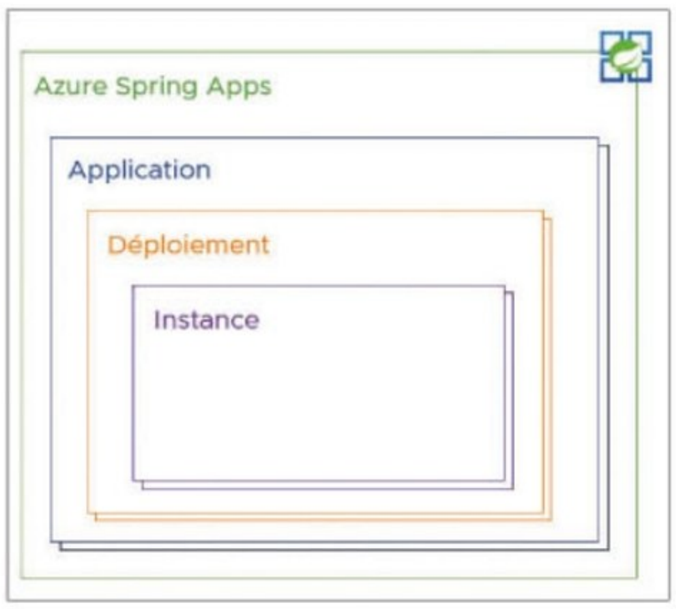


Figure 2
Modèle de déploiement d'Azure Spring Apps

Intégration dans le portail Azure

Azure Spring Apps est intégré au portail Azure ce qui permet aux développeurs d'évoluer dans une interface qu'ils connaissent. Cela leur permet d'avoir une vision complète sur les applications, les déploiements et leurs configurations ainsi que sur tous les composants déjà présents dans Azure.

À partir de ce portail, il est possible de configurer et interagir avec les applications déployées en utilisant les outils mis à disposition dans le portail cela permet notamment de configurer :

- Les règles de configuration applicatives
- Le stockage
- Les sondes de santé (Probe)

AspectJ	Spring Cloud Security	Spring Cloud Stream Binder pour Kafka Streams	Spring Data pour Apache Geode™	Spring Security	Spring Cloud Stream/Task App Starters
Micrometer	Spring Cloud Sleuth	Spring Cloud Stream Binder pour AWS Kinesis	Spring Data Redis	Spring Security OAuth	
Reactor	Spring Cloud Zookeeper	Spring Cloud Vault	Spring Data REST	Spring Security SAML	
Spring AMQP	Spring Cloud Kubernetes	Spring CredHub	Spring Data Solr	Spring Cloud Data Flow	
Spring Batch	Spring Cloud CLI	Spring Data pour Apache Cassandra™	Spring Framework	Spring Session	
Spring Boot	Spring Cloud OpenFeign	Spring Data Commons	Spring HATEOAS	Spring Session pour MongoDB	
Spring Cloud Commons	Spring Cloud Gateway	Spring Data KeyValue	Spring Integration (principaux modules + Kafka + AWS)	Spring Session pour GemFire/Apache Geode™	
Spring Cloud Config	Spring Cloud Function	Spring Data LDAP	Spring pour Apache Kafka®	Spring Tool Suite	
Spring Cloud Netflix	Spring Cloud Contract	Spring Data JDBC	Spring LDAP	Spring Tools 4	
Spring Cloud Bus	Spring Cloud Task	Spring Data JPA	Spring Kerberos	Spring Vault	
Spring Cloud Cloud Foundry	Spring Cloud Stream	Spring Data MongoDB	Spring REST Docs	Spring Cloud Skipper	
Spring Cloud Open Service Broker	Spring Cloud Stream Binder pour Apache Kafka®	Spring Data GemFire	Spring Retry	Spring Shell	
Spring Cloud Consul	Spring Cloud Stream Binder pour RabbitMQ		Spring Statemachine		

Figure 1
Ecosystème Spring supporté officiellement par VMware

- L'identité (un wrapper d'un Service principal) assignée à chaque Application
- Les règles de scaling, etc. **Figure 3**

Il est également possible de se connecter à une instance d'application pour la résolution des problèmes : **Figure 4**

Déploiement

La gestion des environnements est rapide et facilite la pratique de déploiement Blue/Green. ASA autorise deux déploiements pour chaque application, un seul reçoit le trafic de production. Ce modèle est connu sous le nom de déploiement Blue-Green et permet notamment de déployer une nouvelle version d'application sans interruption de service. La prise en charge par Azure Spring Apps du déploiement Blue-Green, avec un pipeline de livraison continue (CD) et des tests automatisés rigoureux, permet des déploiements d'applications agiles avec un niveau de confiance élevé.

Une application peut être déployée avec la CLI, ou à travers un Pipeline de déploiement comme par exemple GitHub Workflow pour lequel des GitHub Actions sont fournies ainsi qu'un Plugin Maven.

Scaling

La mise à l'échelle des applications est prise en charge. Cela peut se faire de manière manuelle en spécifiant un nombre d'instances fixe (avec le niveau Standard, vous pouvez effectuer un scale-out allant jusqu'à 500 instances). Soit de manière automatique en fonction de métriques que vous définissez. **Figure 5**

Supervision

Pour la supervision, vous pouvez analyser les logs avec Elastic ou vous appuyer sur un puit de logs Azure Log Analytics. Log Analytics est intégré à ASA et fournit un certain nombre de requêtes sur étagère, et permet également de réaliser des tableaux de bords customisés. **Figure 6**

ASA supporte des solutions de monitoring tierce comme New Relic, Dynatrace, AppDynamics, Elastic APM et bien sûr Azure Application Insights. Application Insights est complètement intégré à ASA et au portail Azure et ne nécessite aucun paramétrage de la part du développeur.

En utilisant l'agent In-process Java d'Application Insights dans Azure Spring Apps, vous pouvez :

- Affichez une carte des dépendances des applications Spring
- Vérifiez les performances des requêtes
- Surveillez les métriques dynamiques en temps réel
- Vérifiez les échecs des requêtes
- Vérifiez les métriques de l'application
- Consultez les journaux des applications

Intégration avec Spring Cloud Azure

Spring Cloud Azure vous fournit une intégration parfaite dans l'écosystème Azure et vous permets de consommer de nombreux services PaaS :

- Spring Data (PostgreSQL, Maria DB, MongoDB)
- Spring resources (Blob Storage, Azure Files)
- Spring Messaging (Service Bus, Event Hubs)
- Spring Cache (Redis Cache)
- Spring Security (Active Directory, AAD B2C)
- Zero Trust (Key Vault Secrets, Key Vault Certs)

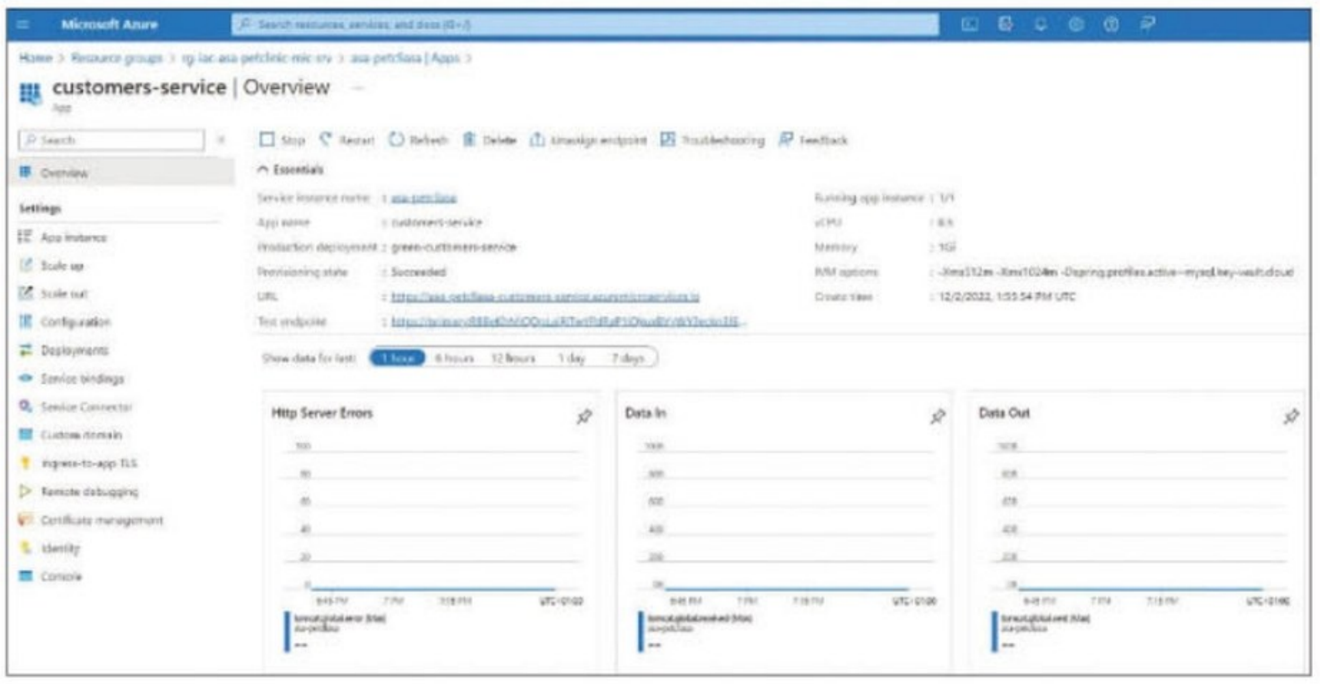


Figure 3 - *Illustration de la vue d'une Application*

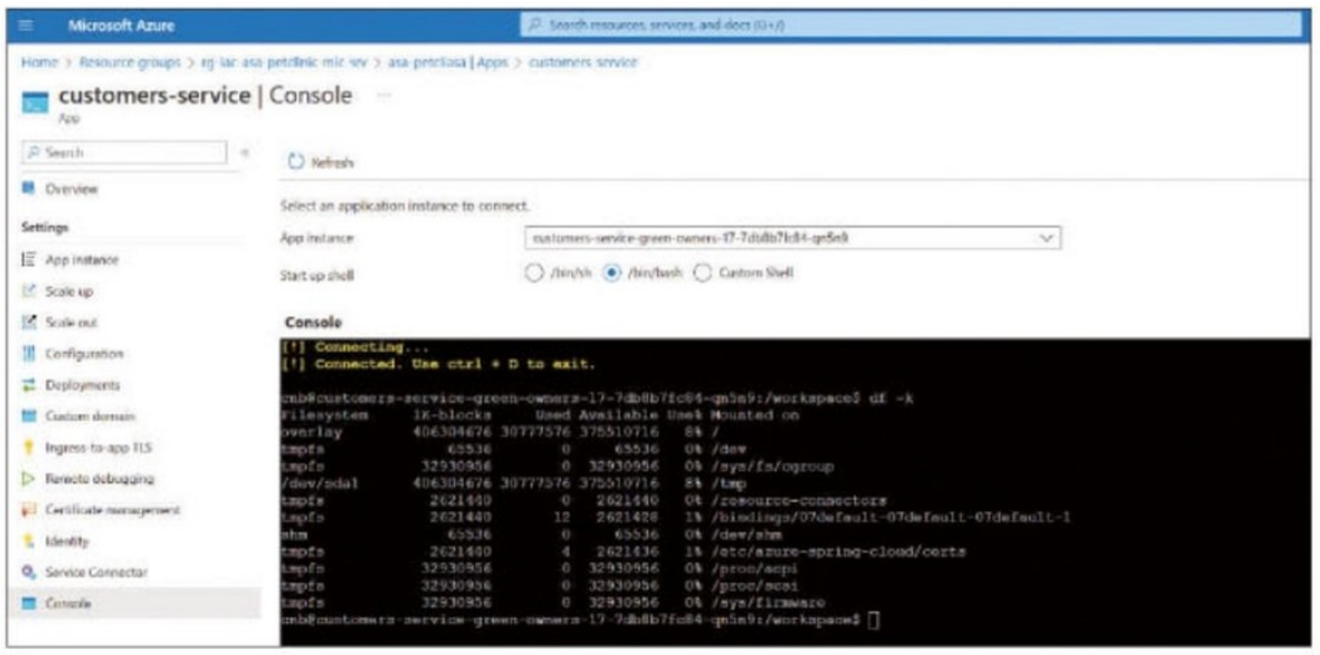


Figure 4 - *Connexion directe sur une instance de l'application pour la résolution des problèmes*

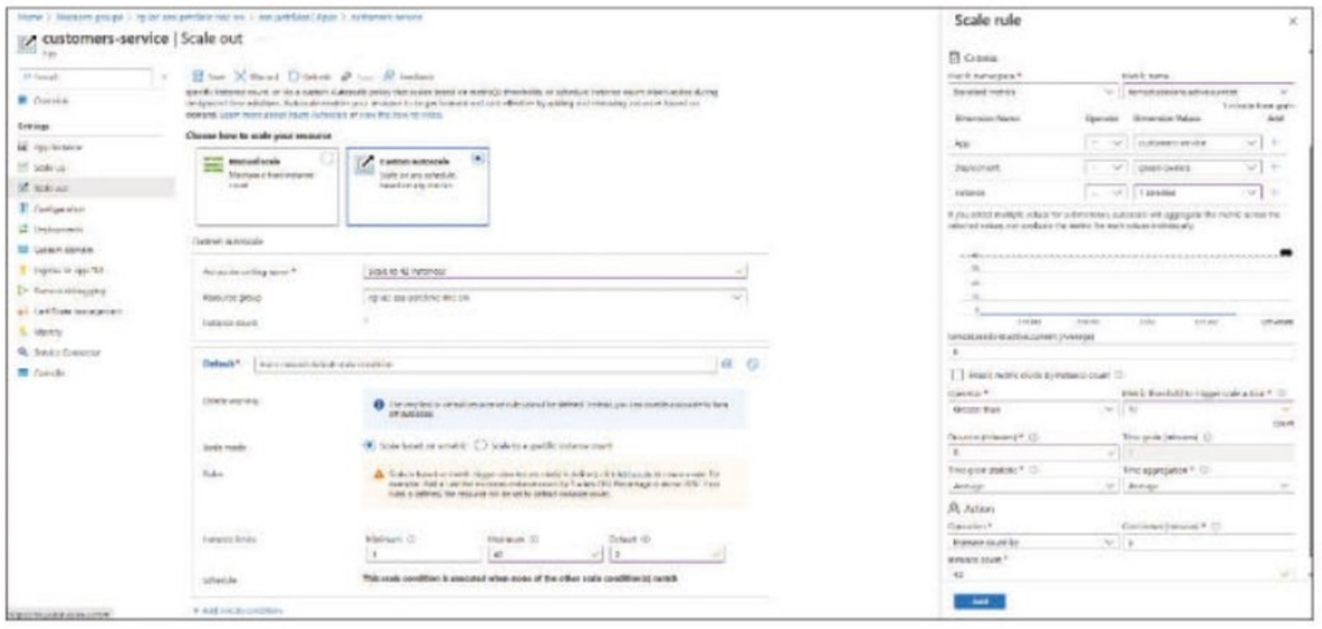


Figure 5 - *Configuration de mise à l'échelle automatique*

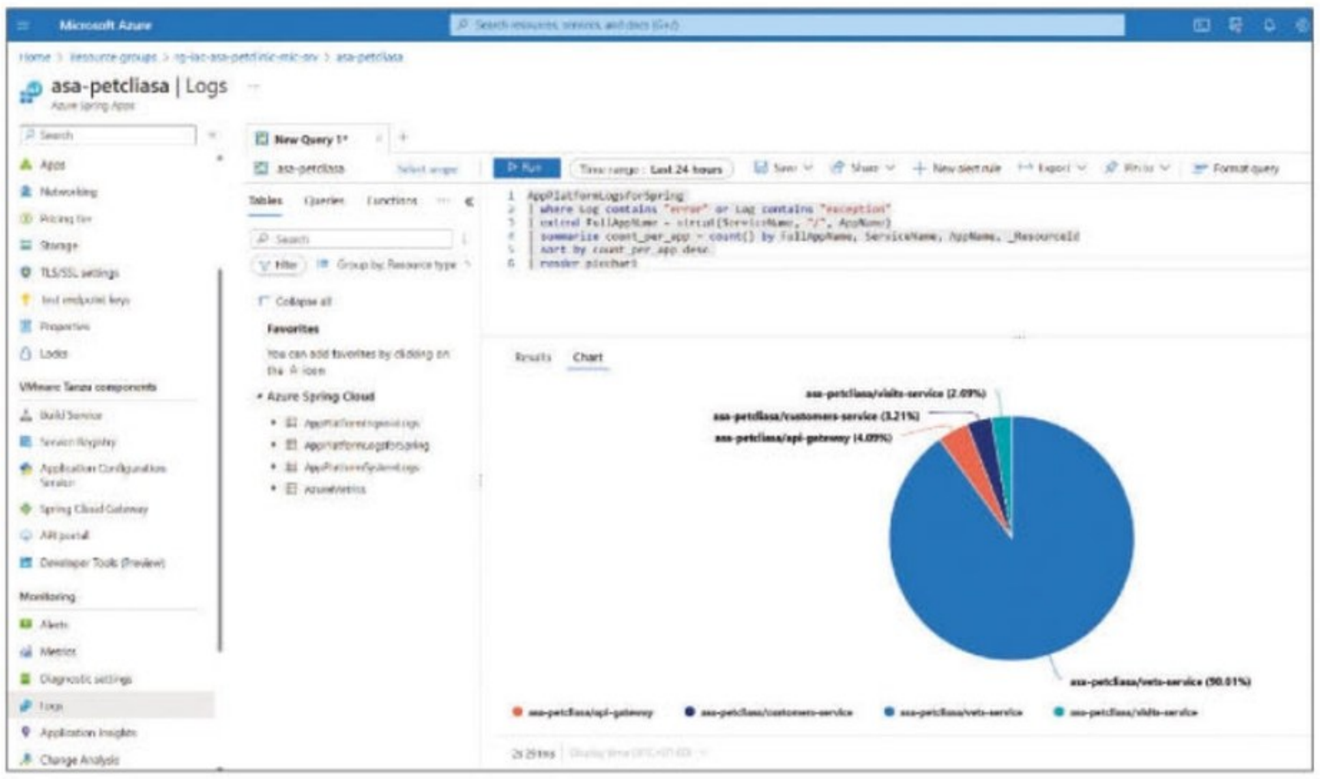


Figure 6 - *Exemple d'un tableau de bord construit sur la base de l'exécution d'une requête de logs*

Prenons un exemple classique nécessitant de devoir gérer un secret : la chaîne de connexion à la base de données. L'utilisateur et l'URL JDBC peuvent être déclarées dans la Configuration globale.

```
spring:
 (...)
 datasource:
  username: pet_user
  url: jdbc:mysql://petclinic.mysql.database.azure.com:3306/petclinic
```

Le mot de passe va lui être récupéré depuis Azure Key-Vault en configurant son endpoint et l'Identité de l'Application habilitée à lire ce secret :

```
spring:
 cloud:
  azure:
   keyvault:
    secret:
     (...)
     property-sources:
      - name: kv-cfg-customers
        endpoint: https://petclinic.vault.azure.net/
       credential:
        managed-identity-enabled: true
        client-id: <your client ID>
```

Exemple de configuration Spring permettant de récupérer les informations du secret

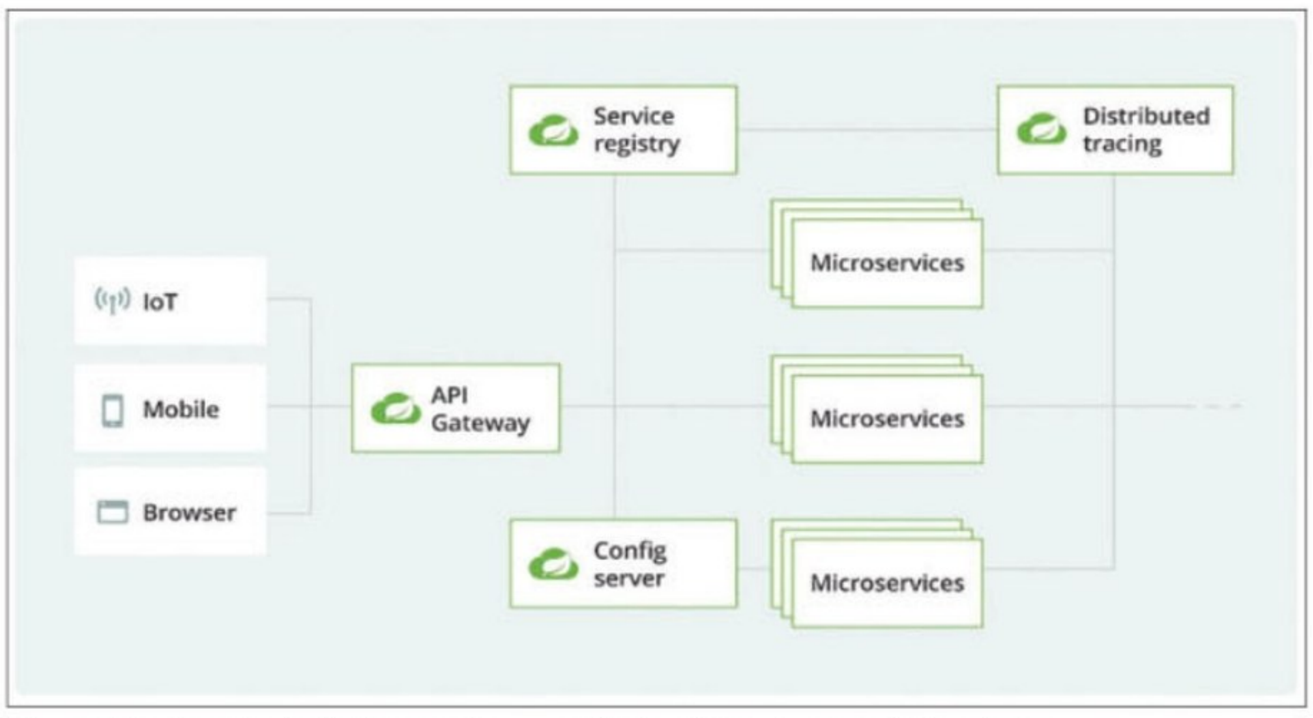


Figure 7 - *Exemple de déploiement d'un ensemble de microservices avec Spring Cloud*

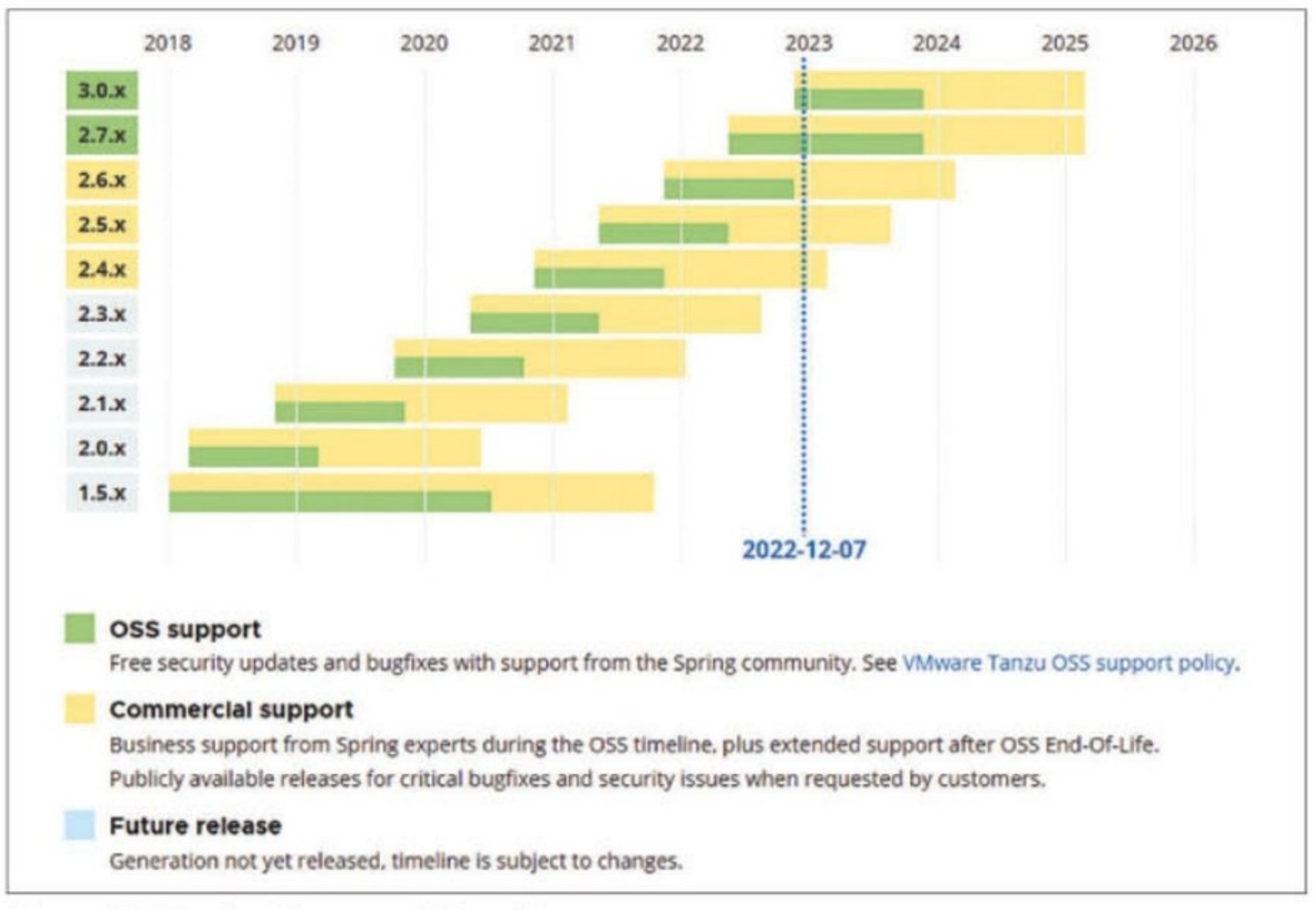


Figure 8 - *Timeline du support de Spring Boot*

INTÉGRATION DE L'ÉCOSYSTÈME SPRING ET DE VMWARE TANZU

Azure Spring Apps Entreprise permet de s'appuyer sur les composants de l'écosystème Spring, nous allons dans un premier temps le présenter, étudier les problématiques qu'il cherche à adresser, avant de présenter en détail les composants mis à disposition et opérés par Microsoft et VMware.

Dans le cadre de cet article, nous allons nous concentrer sur deux briques : Spring Boot et Spring Cloud.

Spring Boot permet de créer des applications basées sur Spring qui sont directement prêtes à l'emploi. Vous pouvez vous rendre sur le site https://start.spring.io/, définir les composants que vous souhaitez utiliser, et vous obtiendrez une application totalement configurée, fonctionnelle et prête à être déployée.

Spring Boot permet entre autres d'adresser la problématique de gestion des compatibilités de versions, d'offrir une configuration par défaut des différents composants, d'avoir des fonctionnalités « production-ready » tels que les endpoints de santé, l'externalisation de la configuration etc.

Cette application sera ensuite déployée de manière autonome, ou bien s'intégrera dans un déploiement de type microservice. La nature distribuée des microservices induit de nouvelles problématiques telles que la communication entre nos applications, la distribution de la configuration, les mécanismes de résiliences etc. Spring Cloud offre un ensemble d'outils permettant d'adresser ces défis, nous allons étudier en détails certains de ses composants mis à disposition dans Azure Spring Apps Entreprise. **Figure 7**

En nous appuyant sur Spring Boot, et Spring Cloud, il est possible de développer et déployer des applications dans une architecture microservice de manière efficiente en nous appuyant sur le savoir-faire et les bonnes pratiques développées depuis plus de 15 ans. Voyons cela en pratique.

VMware Spring Runtime

Lorsque vous déployez vos applications dans Azure Spring Apps Entreprise, vous bénéficiez du support VMware Spring Runtime qui porte sur tout l'écosystème présenté auparavant. En plus d'un accès aux experts Spring 24/7, vous bénéficiez également du support long terme qui étend la durée de maintenance par rapport à la version communautaire.

Grâce à la collaboration entre VMware et Microsoft, vous bénéficiez donc d'un support de bout en bout, de votre code source, jusqu'à son déploiement en production. **Figure 8**

Tanzu Build Service

Le packaging des applications est réalisé via l'utilisation de Tanzu Build Service. C'est une implémentation de la spécifi-

cation Cloud Native Buildpacks de la CNCF (Cloud Native Computing Foundation) qui vise à transformer notre code source en une image de container pouvant tourner dans n'importe quel cloud. Autrement dit, **il n'est pas nécessaire de décrire comment construire un exécutable de notre application**, la construction va se faire automatiquement et à l'état de l'art en s'appuyant sur des composants appelés Buildpacks.
Azure Spring Apps Entreprise ne se cantonne pas à l'écosystème Spring, il est possible d'y construire et déployer des images Java, Dotnet, Go, NodeJS, Python et WebServers.

Application Configuration Service

Une problématique que nous rencontrons lors de nos développements est la gestion des configurations. Les bonnes pratiques de développement nous indiquent qu'il est nécessaire d'avoir une base de code, déployée dans différents environnements, et que la configuration doit être externalisée et spécialisée pour ces différents déploiements.
Il est alors nécessaire de traiter les problématiques de mise à disposition de cette configuration, de synchronisation entre toutes nos instances, de gestion de version etc.
La solution proposée est l'utilisation d'Application Configuration Service. Ce composant nous permet d'avoir un emplacement centralisé pour gérer notre configuration applicative, pour l'ensemble de nos services, sur nos différents environnements. **Figure 9**

Service Registry

La communication entre les différentes applications est primordiale dans un contexte de microservices. Idéalement on souhaiterait qu'un Service A sache communiquer avec un Service B sans que nous ayons à configurer son emplacement. Cela permet d'être beaucoup plus agile et évolutif, en effet si demain ce service change d'url nous aimerions que cette information soit mise à jour en temps réel sans intervention de notre part. C'est ce que permet de réaliser le Service Registry.
Le principe est le suivant : les applications s'enregistrent auprès du Service Registry. Lorsqu'un service veut communiquer avec un autre, il demande à ce Service Registry l'url sur lequel le contacter, le service peut alors le contacter avec cette information. **Figure 10**

Spring Cloud Gateway

Avec la communication par API, il est nécessaire de router les flux vers les bonnes applications qu'elles se trouvent dans Azure Spring Apps, on premise ou autre. Les illusions de l'informatique distribuée mettent en évidence qu'il faut également gérer les problématiques de résilience, de sécurité, de gestion de la bande passante etc. Afin de gérer ces problématiques, nous pouvons nous appuyer sur Spring Cloud Gateway. Via la déclaration d'un ensemble de règles, nous pouvons mettre en œuvre des mécanismes de circuit-breaker, de rate-limiting, des filtres etc. afin de protéger nos applications, et notre système.

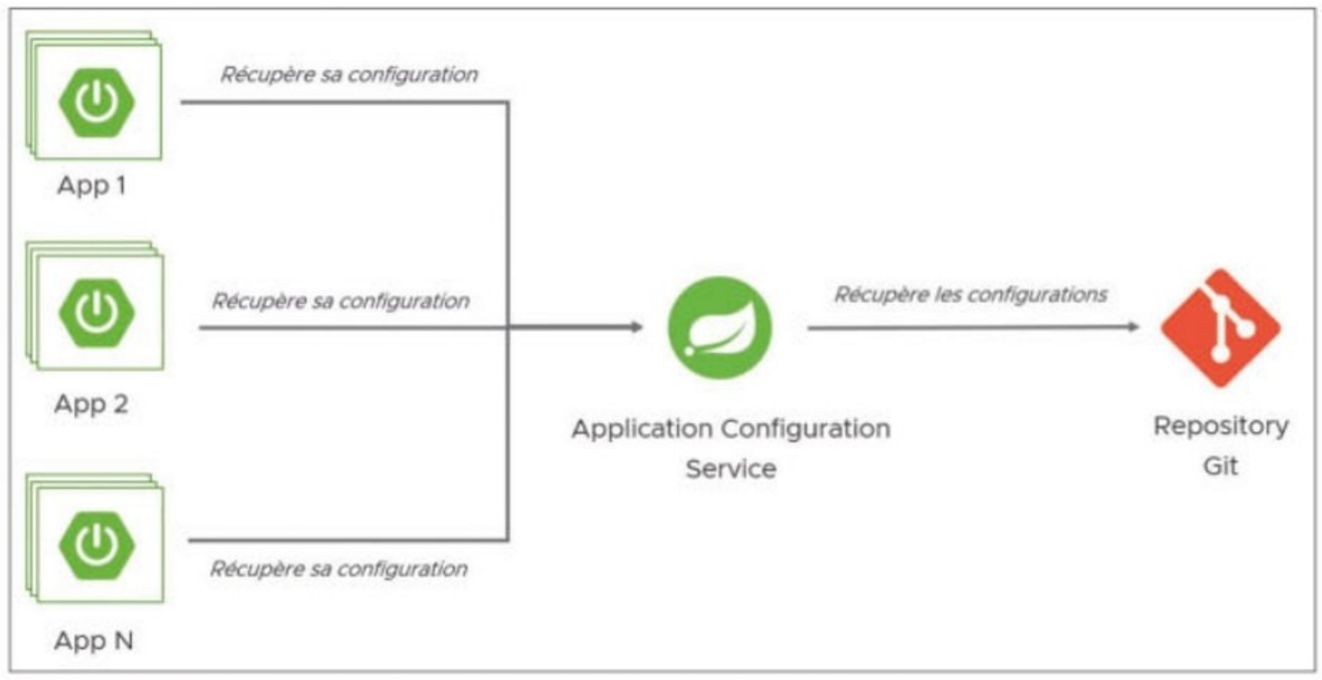


Figure 9 - *Principe de fonctionnement d'Application Configuration Service*

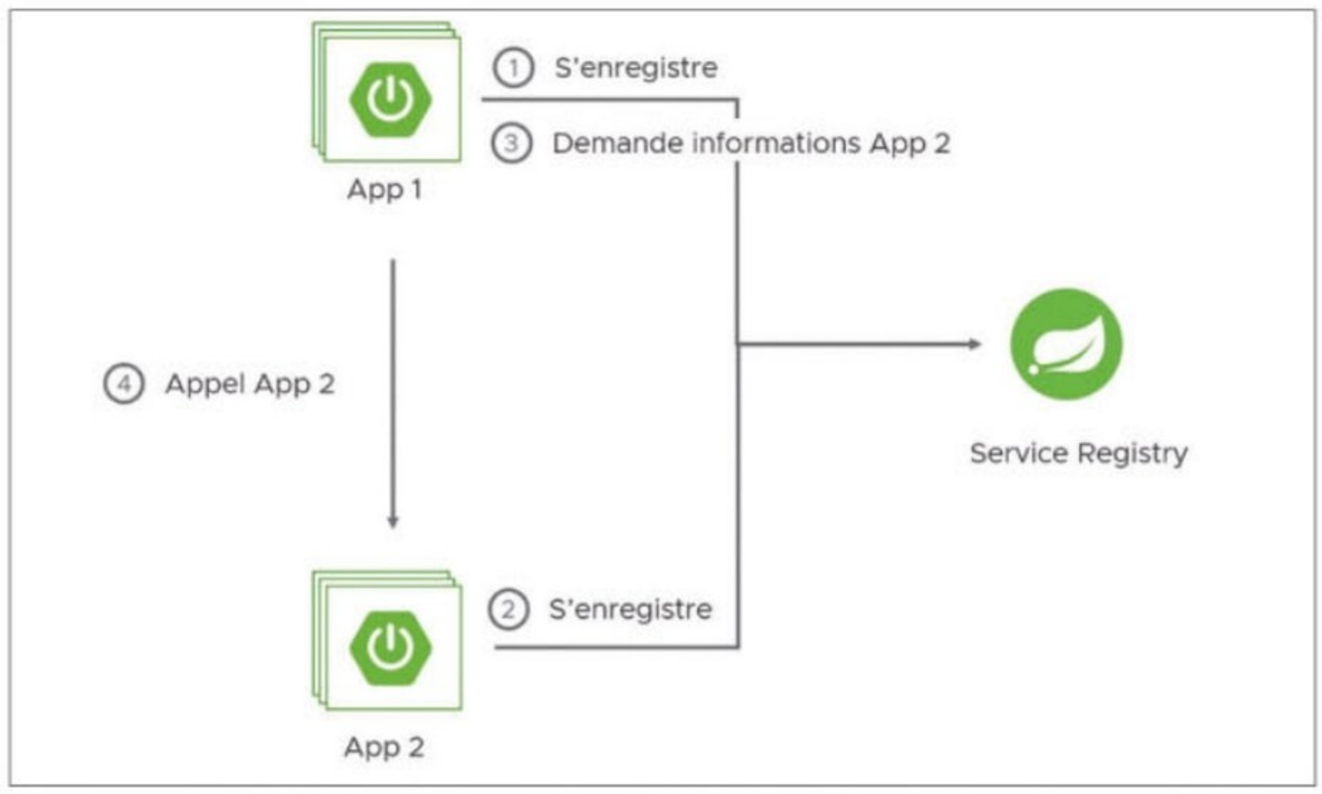


Figure 10 - *Principe de fonctionne du Service Registry*

API Portal

Afin de permettre à nos équipes de développement, mais également à nos clients de pouvoir consommer nos API, un portail est mis à disposition dans Azure Spring Apps Entreprise. Ce portail s'appuie sur la spécification standard du marché : OpenAPI et permet d'avoir accès à la documentation de nos API, mais également de pouvoir les tester. En plus de pouvoir y intégrer des API externes, le portail récupère automatiquement les informations depuis Spring Cloud Gateway, vous avez donc la garantie que votre documentation est alignée avec votre implémentation !

Conclusion

Azure Spring Apps Entreprise est une solution clé en main qui permet aux équipes de se concentrer sur le développement des fonctionnalités. Les différentes solutions mises à disposition permettent de développer et déployer des applications de manière rapide, fiable, et sécurisée avec une disponibilité garantie et du support sur l'ensemble de la chaîne.
Ce n'est que le début de la collaboration entre Microsoft et VMware, de nombreuses nouvelles fonctionnalités sont annoncées telles que la possibilité d'arrêter/relancer votre instance ASA afin d'optimiser les coûts, l'intégration du serverless, les Application Accelerator pour démarrer rapidement avec vos standards d'entreprise ou encore le remote debug depuis IntelliJ et VS Code.

Toujours disponible !

CODER / COMPILER / DÉPLOYER

PROGRAMMEZ !

Le magazine des développeurs

KOTLIN
CLOJURE
LANGAGE COQ
JAVA 20

0 A.D. : LA RÉFÉRENCE DU JEU OPEN SOURCE

Construire un ordinateur 8 bits

Le défi du code

Gaston, 13 ans : « créer à partir du code »

AXA : promouvoir la tech auprès des femmes et des jeunes

N°256
02/03
2023

M 04319 - 256 - F: 6,99 € - RD

® GeorgePeters

TECH CHALLENGER

avec le soutien de AXA

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

Cloud Seed : déployer plus rapidement avec GitLab

Étiez-vous à la Cloud Next (https://cloud.withgoogle.com/next) en octobre dernier ? Google y a organisé son événement annuel où de nouvelles fonctionnalités autour sa plateforme Cloud (https://cloud.google.com) sont présentées. 2022 fut aussi l'occasion de découvrir un nouveau projet de GitLab : Cloud Seed (https://hello.cloudseed.app/).



Jean-Philippe Baconnais

Consultant à Zenika Nantes

GitLab Hero

Community Hero GitPod

Co-orga Human Talks Nantes

https://twitter.com/JPhi_Baconnais

https://nantes.community/

zenika

Qu'est-ce que Cloud Seed ? C'est un projet créé à la fin de 2021 par l'équipe incubatrice de GitLab et dirigée par Sri Rangan (https://twitter.com/srirangan). L'objectif de Cloud Seed est très simple : accélérer le déploiement de vos applications - présentes sur GitLab - vers Google Cloud Platform.

Ce projet a tout d'abord été ouvert en *bêta testing*. J'ai eu l'opportunité de faire partie de ces 100 personnes qui ont pu tester Cloud Seed et donner leur feedback en avant-première.

À la Cloud Next, Sri a annoncé la première release de Cloud Seed ! L'occasion de sortir ce projet de la phase de *bêta testing* et de le mettre à disposition de tous.

La présentation est disponible sur cette page : https://cloud.withgoogle.com/next/speakers?session=PAR21

Avant Cloud Seed, les déploiements ressemblaient à ceci…

Avant Cloud Seed, les déploiements sur Google Cloud pouvaient être réalisés de plusieurs façons. Dans tous les cas, l'intégration et le déploiement continus de vos applications stockées dans GitLab sont réalisés avec un outil fait maison : **GitLab CI** (https://docs.gitlab.com/ee/ci/ CI). Écrits en Go, les **GitLab Runners** (https://docs.gitlab.com/runner/) permettent d'exécuter très rapidement ces opérations de CI/CD définis dans des **pipelines**.

Il y a deux concepts à connaître quand on évoque GitLab CI : les "**stages**" et les "**jobs**".

- un "**stage**" est une étape de votre pipeline, par exemple "test", "build", "deploy", pour tester, construire et déployer votre application.
- un "**job**" est une entité très petite, qui va permettre de réaliser une opération sur votre application. Plusieurs jobs peuvent être présents dans le même "**stage**".

La création d'un pipeline GitLab CI se fait dans un fichier de référence **.gitlab-ci.yml**.

Cette cheat sheet (https://dev.to/zenika/gitlab-cheatsheet-1-basics-of-stages-and-jobs-14p0) que j'ai créée explique les bases des pipelines GitLab CI : **figure 1**.

Retournons sur nos déploiements vers le Cloud. Je vais me focaliser uniquement sur des solutions basées sur GitLab CI. En excluant volontairement des outils externes comme Ansible (https://www.ansible.com/) et Terraform (https://www.terraform.io/) qui peuvent être exécutés à partir d'un pipeline GitLab CI, je retiens deux options pour déployer un projet sur GCP.

La première consiste à utiliser l'outil de ligne de commande ("*Command Line Interface*" - CLI) de Google, **gcloud** (https://cloud.google.com/sdk/docs/install), que vous utilisez probablement déjà sur votre ordinateur.

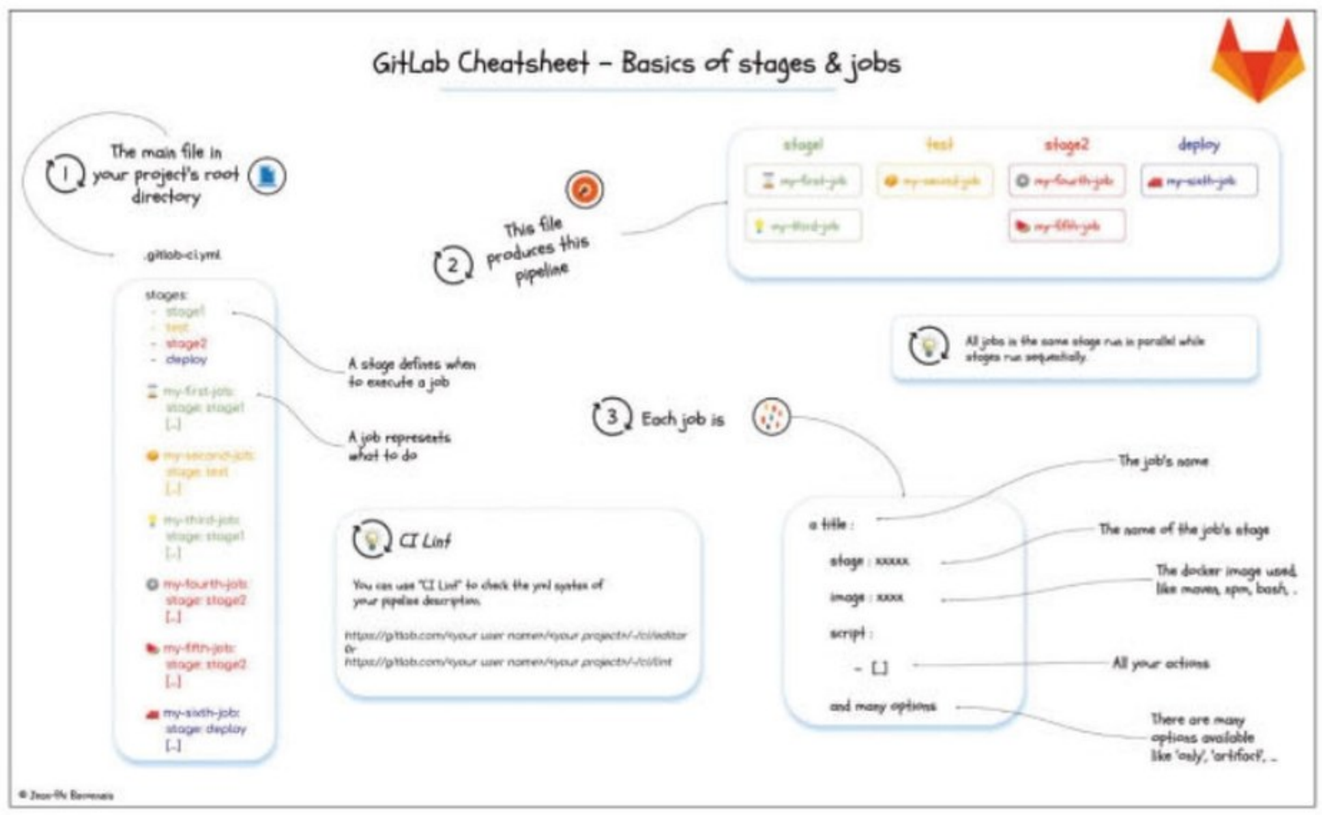
À partir d'un nouveau **job** se basant sur l'image **google/cloud-sdk**, vous allez accéder à un environnement mettant à disposition l'outil **gcloud**. Après avoir configuré la partie authentification de votre projet, vous pouvez déployer rapidement avec **gcloud** une **Cloud Function** (https://cloud.google.com/functions), une application avec **App engine** (https://cloud.google.com/appengine) ou d'autres services Google. Ce job pourrait ressembler à ceci :

```
deploy-with-gcloud-image:
 [...]
 image: google/cloud-sdk:<version>
 script:
  - echo $GCP_SERVICE_KEY | base64 -d > /tmp/gcloud-service-key.json
  - gcloud auth activate-service-account —key-file /tmp/gcloud-service-key.json
  - gcloud functions deploy my_function [...]
```

Si vous travaillez sur un projet plus conséquent, déployé sur **Google Kubernetes Engine** (https://cloud.google.com/kubernetes-engine), vous pouvez utiliser l'agent Kubernetes (https://docs.gitlab.com/ee/user/clusters/agent/) créé par GitLab il y a un an environ. Cet agent a remplacé une intégration Kubernetes plus ancienne, dépréciée dans la version 14.5 de GitLab et supprimée dans la version 15 sortie en 2021.

Comment l'agent fonctionne-t-il ? Vous configurez un agent sur votre projet GitLab. Cet agent est représenté par un ou plusieurs fichiers de configuration yaml. Après avoir déclaré

Figure 1



dans l'interface de GitLab où se trouvait votre agent, GitLab vous donne une commande qui vous permettra d'installer cet agent sur votre cluster Kubernetes. Et c'est tout ! Après cela, chaque commit sur votre projet sera détecté par l'agent et votre application sera déployée automatiquement. Top !
Si malgré cela, vous voulez ou devez déployer des éléments sur votre cluster Kubernetes avec GitLab CI, cet agent vous offre la possibilité d'utiliser le "tunnel de CI/CD" (https://about.gitlab.com/blog/2022/01/07/gitops-with-gitlab-using-ci-cd/). Ce tunnel consiste à vous simplifier la connexion entre GitLab et votre cluster Kubernetes en déclarant uniquement votre agent. Dans un job basé sur l'image **bitnami/kubectl**, la déclaration de votre contexte Kubernetes avec votre agent vous suffira pour appliquer des fichiers yaml de configuration Kubernetes avec l'outil **kubectl** (https://kubernetes.io/fr/docs/tasks/tools/install-kubectl/). Par exemple :

```
deploy-with-cicd-tunnel:
 before_script:
  - kubectl config use-context "group/project:agentk"
 image:
  name: bitnami/kubectl:<version>
  entrypoint: [""]
 script:
  - kubectl apply -f <my kubernetes file>
```

L'intégration Kubernetes est expliquée dans cette **cheat sheet** (https://dev.to/zenika/gitlabcheatsheet-12-kubernetes-integration-2ph2) : **figure 2**.

Figure 2



Cloud Seed : le facilitateur de déploiement

L'objectif principal de Cloud Seed est d'accélérer le déploiement de vos apps. Dans le précédent paragraphe, vous avez pu voir que la déclaration et rédaction d'un pipeline était une étape obligatoire pour pouvoir déployer votre application. Rien de vraiment compliqué, mais cela reste une tâche à réaliser et qui est généralement copiée-collée d'autres projets.
Cloud Seed apporte également d'autres objectifs "indirects". Si le déploiement de vos applications dans le Cloud est plus rapide et plus facile, cela permettra d'attirer et de convaincre d'autres personnes de migrer vers le Cloud. Ensuite, si ces applications sont déployées sur le Cloud, elles pourront profiter des avantages des plateformes Cloud, leur apportant de la modernité.

Comment Cloud Seed fonctionne ?

Cloud Seed est disponible dans GitLab à partir du menu "Infrastructure > Google Cloud". Cela amène à voir une nouvelle page contenant trois parties : "Configuration", "Deployments" et "Database" : **figure 3**.

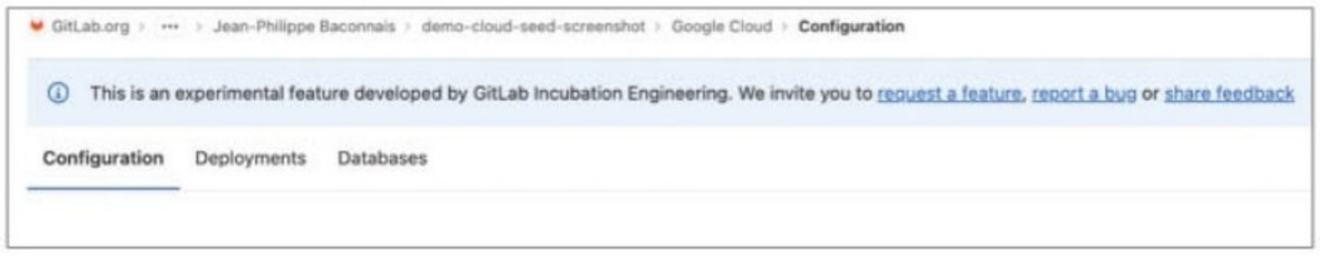
Configuration

Cette première section, obligatoire, permet de configurer votre connexion à votre compte Google. Après avoir rentré et choisi votre compte Google, vous pouvez sélectionner votre projet GCP préalablement créé. Ensuite, vous spécifiez sur quelle branche Git vous voulez créer un compte de service : **figure 4**

Note : *pendant mes tests, je n'ai pas essayé de créer des éléments sur une branche spécifique. Ce sera pour plus tard.*

Après cette étape, vous verrez votre nouveau compte de service affiché à l'écran et dans votre projet GCP. La dernière chose à configurer est la région GCP. La région par défaut est "us-central1".
Ces opérations impactent vos variables de CI/CD. En vérifiant dans la partie "Settings > CI/CD > Variables", vous verrez une liste de variables configurées pour vous. Avant Cloud Seed, ces déclarations étaient à faire manuellement.

Figure 3



Déploiements

Maintenant que la configuration est effective, nous allons pouvoir se concentrer sur l'objectif principal de Cloud Seed : le déploiement de votre application.

Au moment d'écrire cet article, il y a deux services GCP visibles dans GitLab : **Cloud Run** (https://cloud.google.com/run?hl=fr) et **Cloud Storage** (https://cloud.google.com/storage). Seul **Cloud Run** est activé pour le moment.

Cloud Run est une plateforme **serverless** qui permet de se concentrer sur le développement. **Cloud Run** construit pour vous votre application à partir d'un Dockerfile ou, si ce fichier est manquant, essaye de déterminer la typologie de votre projet à partir d'une liste de **buildpacks** (https://github.com/GoogleCloudPlatform/buildpacks).

Cloud Seed vous permet d'intégrer **Cloud Run** à votre projet simplement avec une **merge request** contenant une modification sur le fichier **.gitlab-ci.yml**.

```
include:
- remote: https://gitlab.com/gitlab-org/incubation-engineering/five-minute-production/library/-/raw/main/gcp/cloud-run.gitlab-ci.yml
```

Cette modification crée un lien avec un autre projet GitLab, appartenant à GitLab, contenant des templates de CI/CD pour les différents services GCP.

En regardant ce fichier yaml dans le détail, nous pouvons voir que le job est assez simple :

```
deploy-to-cloud-run:
 stage: deploy
 image: registry.gitlab.com/gitlab-org/incubation-engineering/five-minute-production/library/google-cloud-sdk-for-gitlab:main
 script:
  - cp /cloud-run.sh .
  - ./cloud-run.sh
[...]
```

Le job inclus dans votre projet sera exécuté dans le stage **deploy** dans un job appelé **deploy-to-cloud-run**. Ce job exécutera le script **cloud-run.sh**, sauvegardé dans le même projet. C'est un script shell qui vérifie la présence de toutes les variables nécessaires et exécute une commande **gcloud** telle que vous le feriez sur votre ordinateur ou dans votre CI.

Et c'est tout !

Si vous avez un fichier **Dockerfile** ou travaillez sur un projet dont sa typologie apparaît dans la liste des **Google buildpacks**, votre application sera déployée dans **Cloud Run** une fois ce job terminé.

Si vous avez un doute, vous pouvez voir dans la console GCP ou avec la CLI **gcloud** que vous avez bien un service **Cloud Run** en cours d'exécution : **figure 5**

L'application, dans mon exemple une API Quarkus, est bien disponible ! : **figure 6**

Dans le job **deploy-to-cloud-run**, il y a la notion d'environnement que j'ai précédemment et volontairement mis de côté :

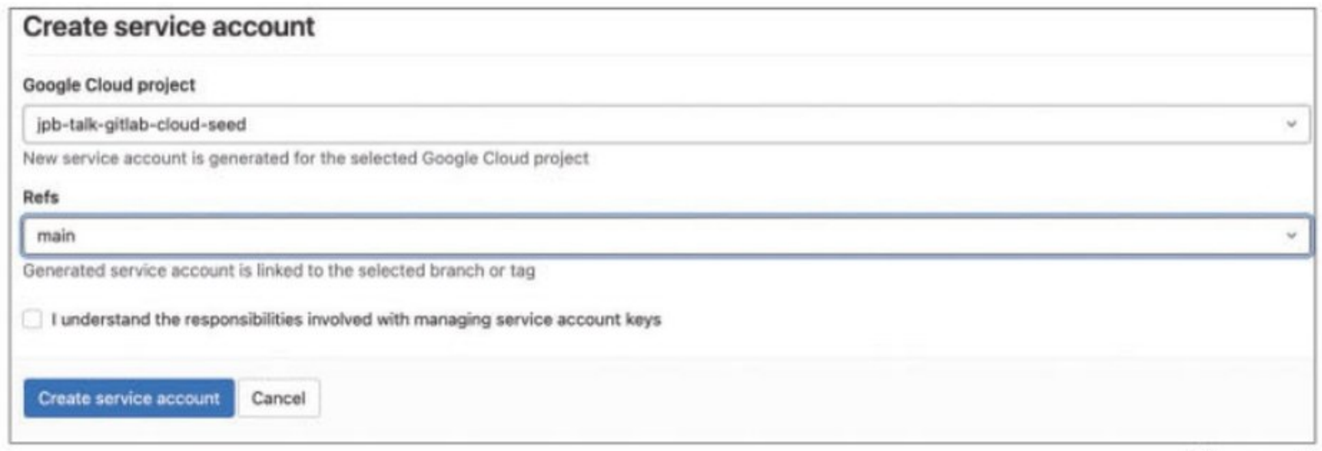


Figure 4



Figure 5

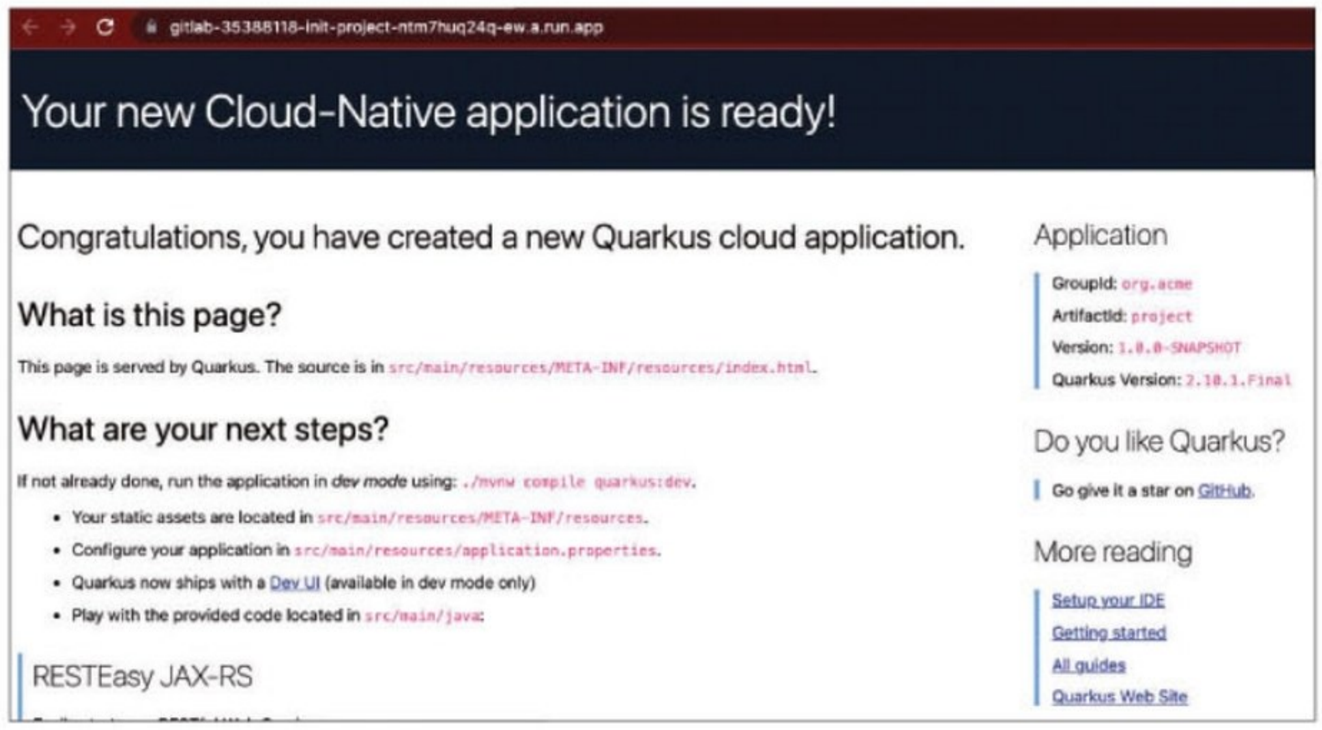


Figure 6

```
environment:
  name: $CI_COMMIT_REF_NAME
  url: $DYNAMIC_URL
```

Ces trois lignes vont permettre à **Cloud Seed** de créer un service **Cloud Run** par branche. Très pratique pour vérifier vos développements en cours.

Dans GitLab, vous pouvez visualiser ces environnements : **figure 7**

Un environnement, dans le sens GitLab, n'a pas d'action directe sur GCP. GitLab affiche les informations et vous permet d'accéder à l'URL accessible dans GCP.

Dans mon exemple (**figure 7**), trois branches sont présentes, deux possèdent l'intégration avec **Cloud Run**. Le bouton **Open** permet d'accéder à l'URL de votre service **Cloud Run**.

Pour le moment, le bouton "Stop" n'a pas d'impact sur GCP. Votre service **Cloud Run** ne sera donc pas supprimé. Il permet seulement de supprimer l'environnement visible dans GitLab.

Bases de données

Cloud Seed permet également de créer des bases de données et d'afficher leurs informations directement dans GitLab. Actuellement, la création d'instance de bases de données Cloud SQL (https://cloud.google.com/sql) est possible en quelques clics pour Postgres, MySQL et SQL Server.

Pour créer une base de données Postgres à partir de **Cloud**



Figure 7

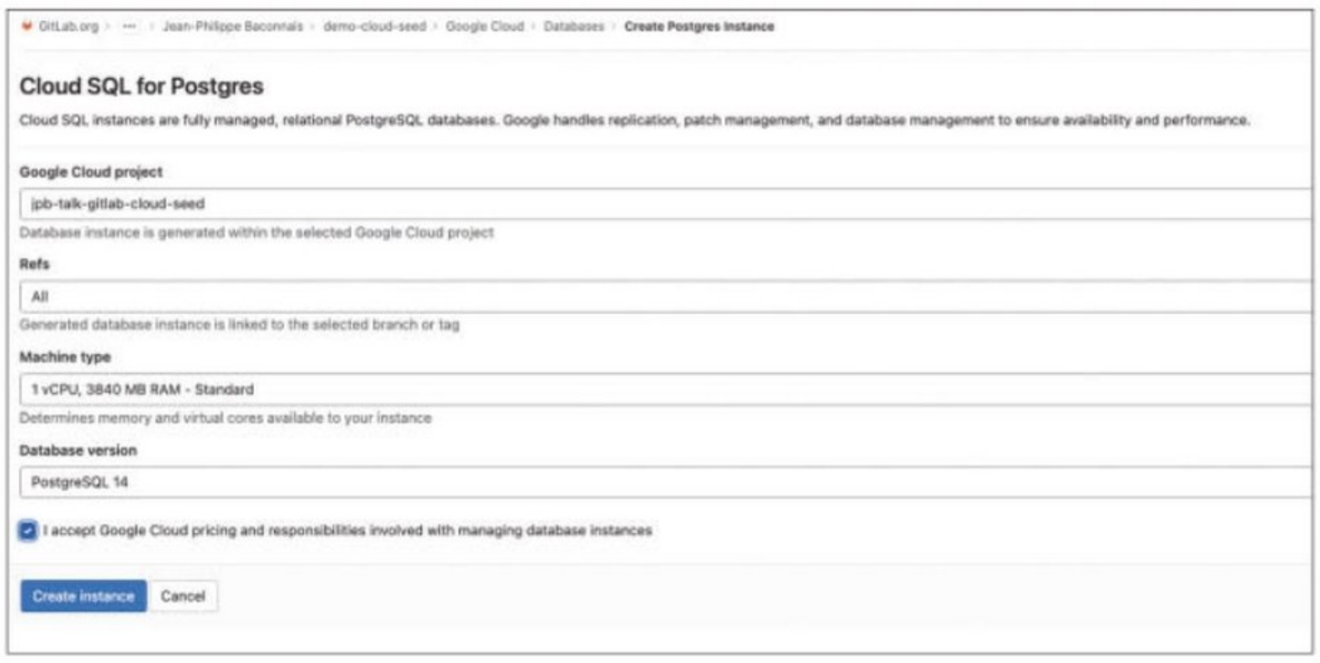


Figure 8

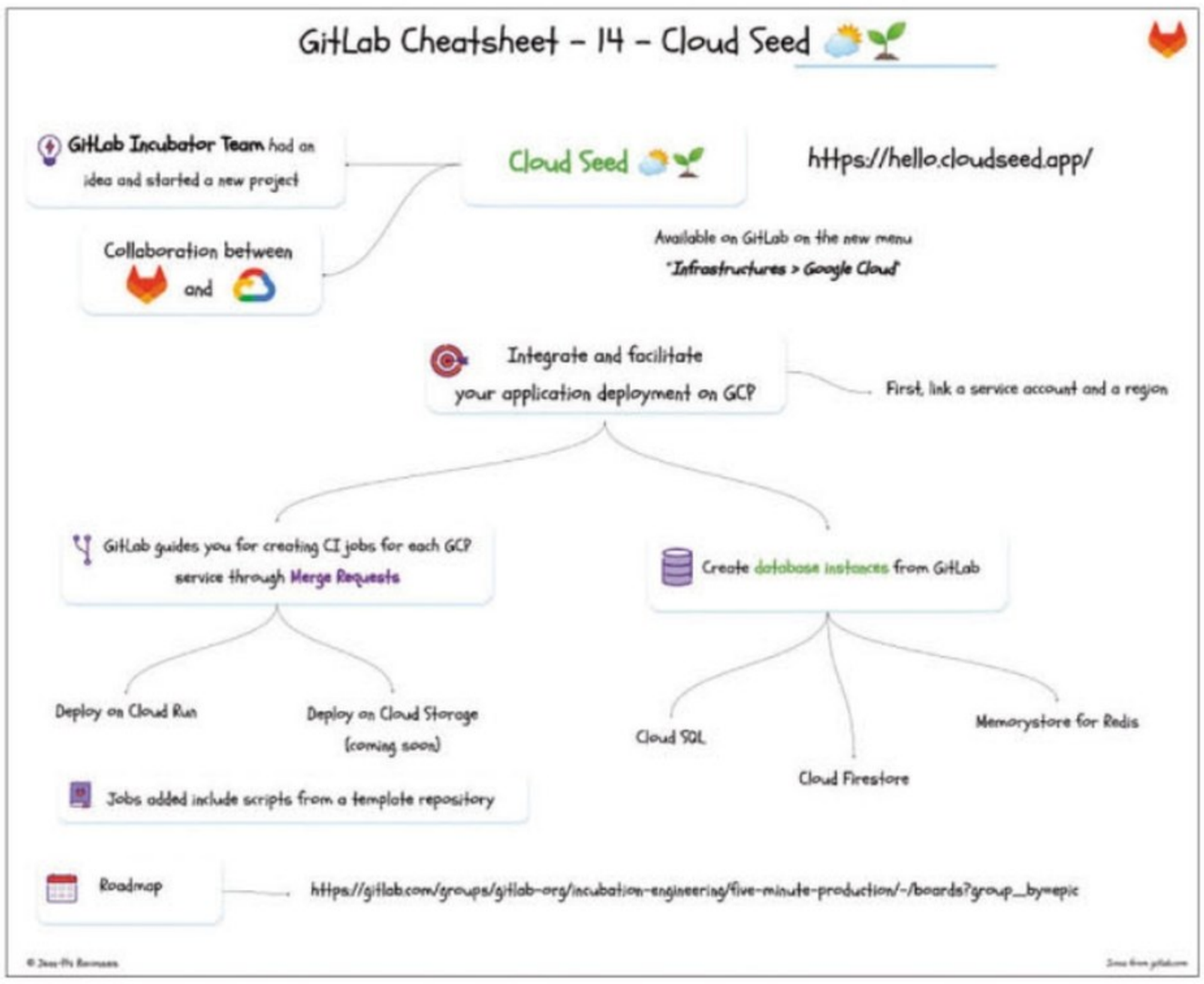


Figure 11

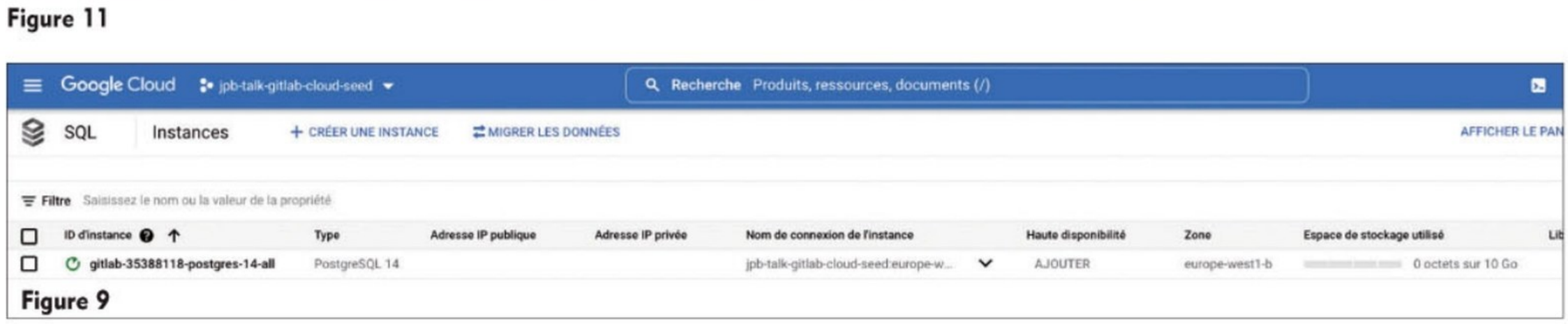


Figure 9

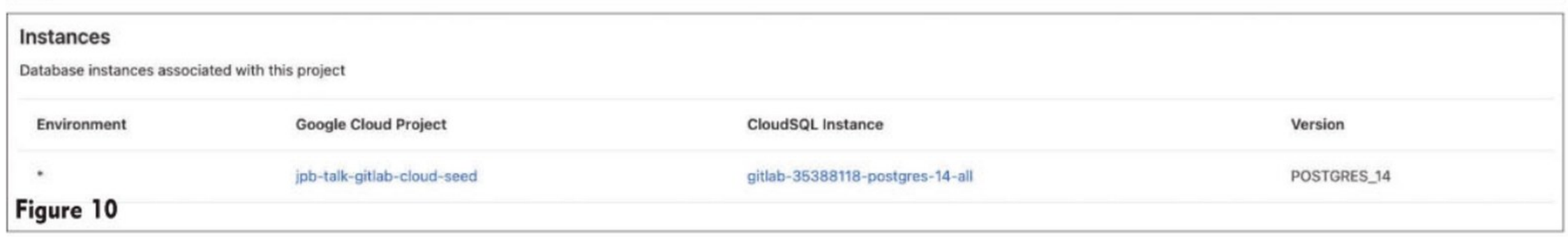


Figure 10

Seed, il suffit d'inscrire le nom, le type de machine, la version de Postgres et une référence de branche si l'on veut créer une instance de base de données pour cette branche uniquement : **figure 8**.

Après cette étape, vous verrez bien entendu cette nouvelle instance sur la console GCP ou via la CLI **gcloud** : **figure 9**. Actuellement, seules ces informations sont disponibles sur GitLab : **figure 10**. Dans une issue que j'ai créée, je soumets l'idée d'en afficher plus .

Comme tous les projets GitLab, le plan de route de **Cloud Seed** est accessible à tout le monde à partir de ce lien https://gitlab.com/groups/gitlab-org/incubation-engineering/five-minute-production/-/boards.

Le futur de Cloud Seed

Après avoir pu tester ces premières fonctionnalités, je suis content d'avoir pu constater et profiter de la rapidité et de la simplicité avec laquelle **Cloud Seed** nous permet de déployer des applications sur GCP sans avoir à écrire et à configurer la connexion avec GCP dans des jobs de CI/CD.

Cette étape n'est pas la plus difficile. Je le fais pour tester des services. Lorsque je développe, je souhaite me concentrer sur le développement.

Sur **Cloud Seed** et comme sur la majorité des projets GitLab, les feedbacks sont les bienvenus, sont pris en compte et permettent d'échanger et de participer à la roadmap de ce projet. Les contributions sont les bienvenues !

Depuis ces premiers tests, j'ai décidé d'intégrer **Cloud Seed** sur d'autres projets et attends avec impatience l'arrivée des prochaines intégrations de services GCP.

Pour expliquer rapidement ce qu'est Cloud Seed, j'ai créé cette 14e **cheat sheet** GitLab (https://dev.to/zenika/gitlab-cheat-sheet-14-cloud-seed-3oh7) : **figure 11**.

J'ai eu la chance de pouvoir présenter Cloud Seed au DevFest Nantes 2022 (https://devfest.gdgnantes.com/). La vidéo est disponible ici : https://www.youtube.com/watch?v=d06ASsadHqM.

Si vous avez des questions ou des commentaires à propos de cet article ou de Cloud Seed, vous pouvez me contacter sur Twitter (@JPhi_Baconnais), pinguer directement le compte Twitter de Cloud Seed (@OpenCloudSeed) ou venir sur le Discord GitLab où vous trouverez un canal #cloud-seed : https://discord.gg/gitlab.

DEVeloppez-vous
avec les Editions ENI



Plus de 200 livres sur le développement informatique

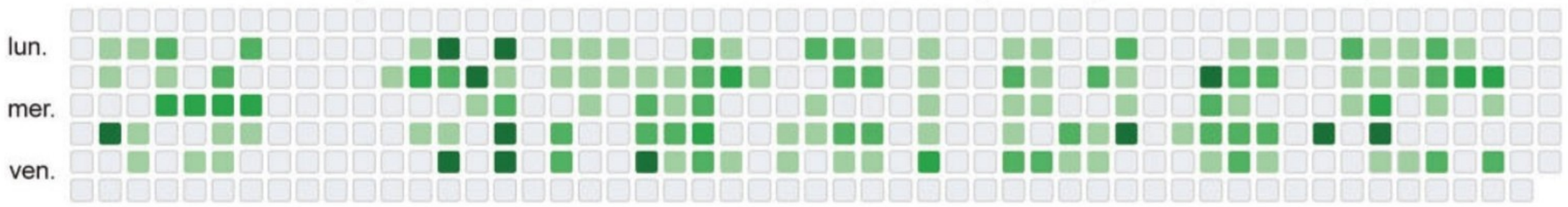
pour les particuliers
editions-eni.fr

pour les entreprises
eni-elearning.com

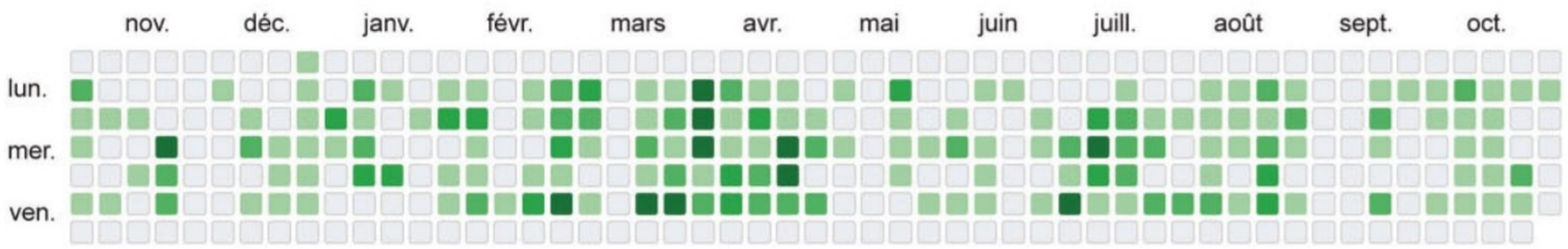
dans toutes les
bonnes librairies

CHEZ SOCIÉTÉ GÉNÉRALE LA CONTRIBUTION EST UNE VALEUR QUE L'ON PARTAGE

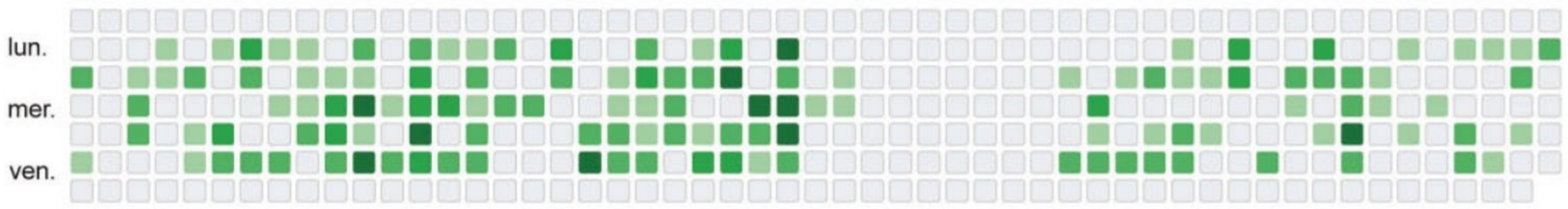
Sophie



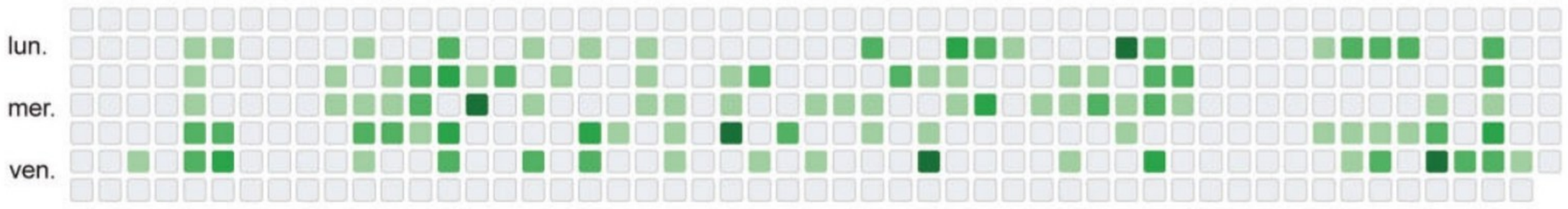
Sébastien



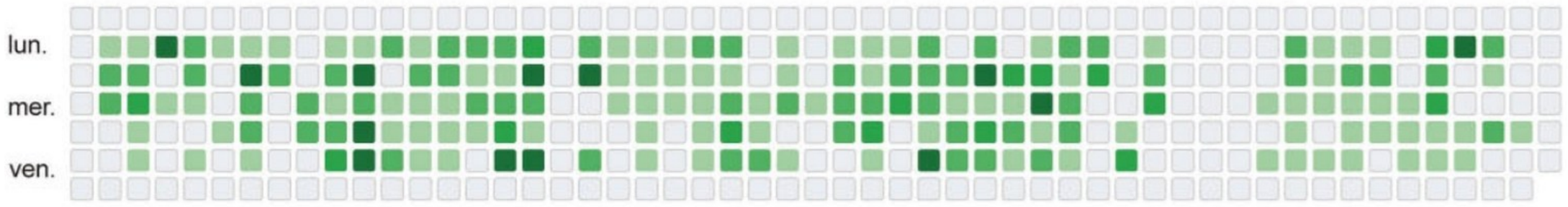
Loïc



Abdel



Yasmine



L'équipe IT recrute !
CDI, alternance, stages
Découvrez nos offres

C'EST VOUS L'AVENIR

SOCIETE GENERALE

L'image des calendriers des contributions est utilisée avec l'autorisation de GitHub, Inc.
Société Générale, S.A. au capital de 1 062 354 722,50 € – 552 120 222 RCS PARIS – Siège social : 29, bd Haussmann, 75009 PARIS. Mars 2023.