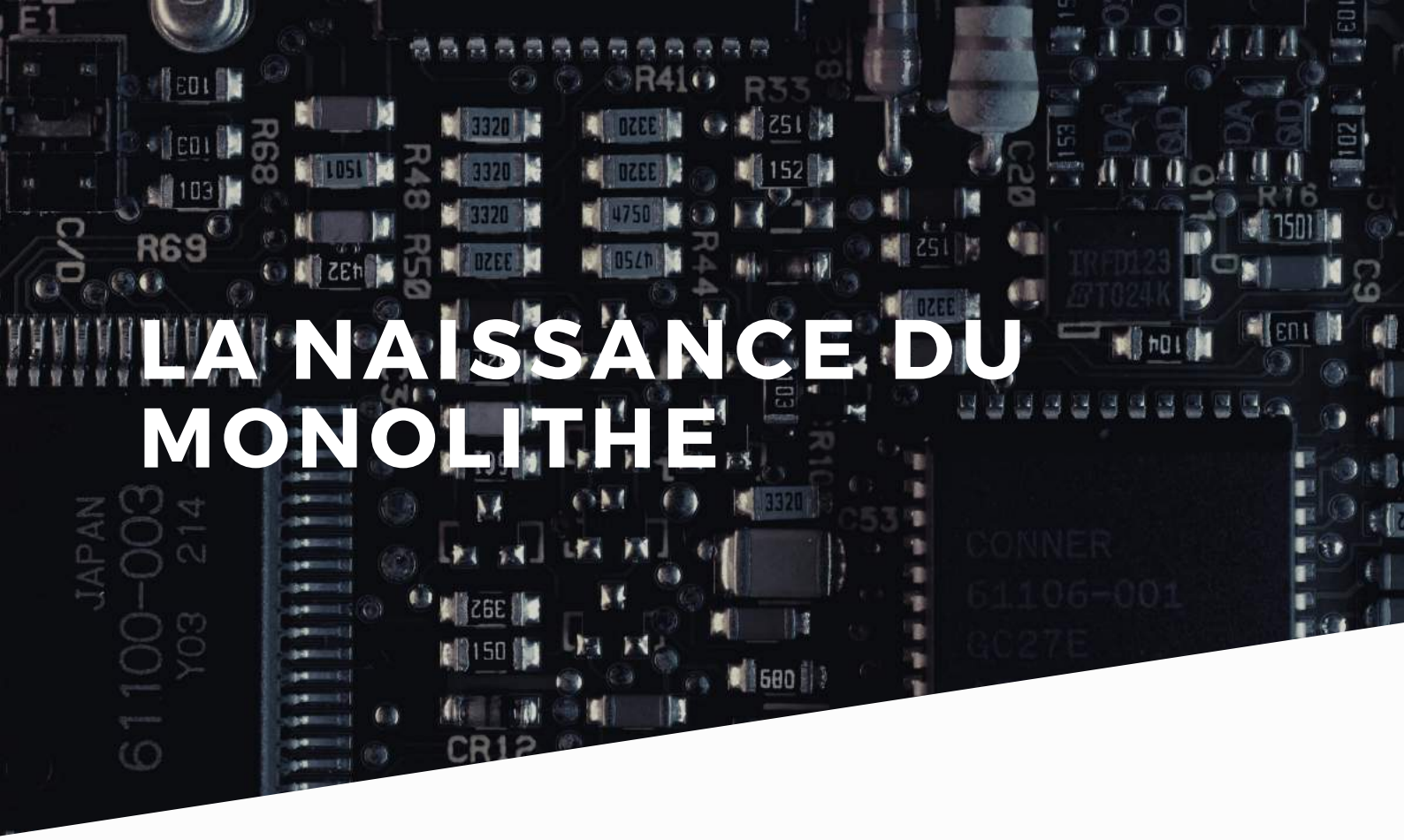
LES ÉVOLUTIONS DU SI

mind 7
CONSULTING

SOMMAIRE



LA NAISSANCE DU MONOLITHE

Savez-vous ce qu'est un monolithe ?

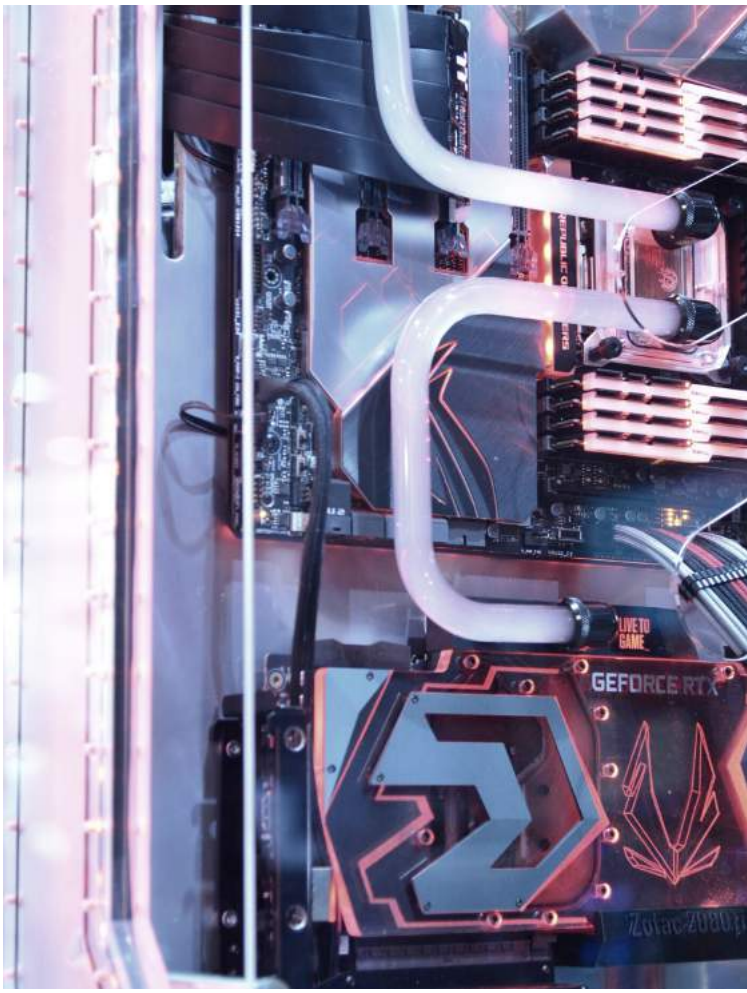
Ne vous méprenez pas, nous parlons bien évidemment du monde de l'IT et non de 2001, l'Odyssée de l'espace ! (by the way, si vous ne l'avez pas vu… vous devriez !).
Pour comprendre pourquoi on parle depuis quelques années maintenant des applications « monolithes », il faut regarder un peu les évolutions des Systèmes d'Information.

ÉTAPE 1 : LA CRÉATION DU MONOLITHE

Historiquement, avant l'an 2000, des nombreuses organisations ont mis en place des applications. Ces dernières visaient à traiter leurs flux croissants d'information, et au passage mettre sous contrôle leurs processus administratifs, industriels et commerciaux. Souvent, une des motivations était le fameux « Bug de l'an 2000 » qui promettait de nous ramener à l'âge de pierre dès le 1er janvier. Mais c'est une autre histoire !

Dans ce contexte, chaque métier, chaque BU, chaque population d'utilisateur connaissait son propre métier. Chacun pouvait définir ses processus et mettre en place un applicatif répondant à ses attentes et besoins. Très vite le partage de données devenait une question cruciale. Comment l'équipe Commerciale peut-elle partager ses prévisions de commandes avec l'équipe de Production ?

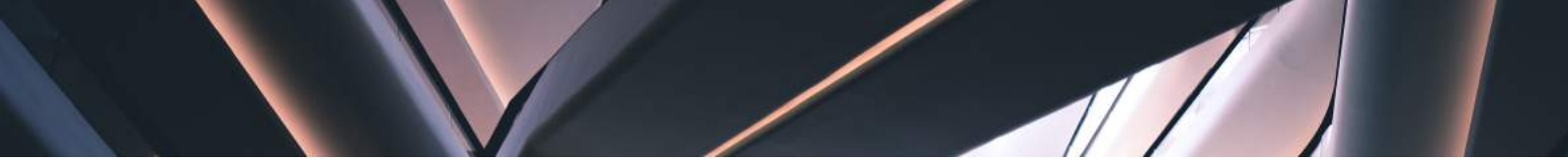
La première réponse a été de mettre en place des ERP (Enterprise Ressource Planning en anglais – PGI ou Progiciel de Gestion Intégrée en français). C'est-à-dire une seule et même application pour tout le monde (ou presque). Du coup, c'est simple, une seule énorme base de données et tout le monde partage la même information. Tout le monde travaille ensemble dans le monde merveilleux des Bisounours.



Cela a amené tout un tas d'autres questions sur la qualité des données et des chantiers entiers de Data gouvernance, mais laissons ça pour une autre fois !

Le fait est qu'à la fin du siècle dernier, l'habitude était de mettre en place un monolithe pour répondre à chacun des besoins. Ces monolithes pouvaient avoir un périmètre plus ou moins étendu (toute l'entreprise ou seulement un groupe d'utilisateurs).

Toujours est-il qu'ils étaient un monolithe, dans le sens où ils traitaient complètement plusieurs étapes d'un ou plusieurs processus, pour des utilisateurs donnés et géraient leur propre ensemble de données.



UN EXEMPLE POUR BIEN COMPRENDRE

Prenons l'exemple des projets SAP ou des applicatifs Achat qui ont été mis en place à cette époque… À titre d'exemple, dans une solution d'e-procurement, on retrouvait toutes les fonctions requises :

- Création, modification, suppression d'un fournisseur
- Création, modification, suppression d'un catalogue achat (liste d'article avec les conditions d'achat)
- Workflow pour l'ensemble du processus
- Création, modification, suppression d'une demande d'achat
- Création, modification, suppression d'une commande d'achat

Chaque applicatif monolithique était développé et géré par une équipe dédiée et pour des utilisateurs dédiés. C'est ça le monolithe ; et il répond à sa propre problématique métier (la gestion des stocks, le traitement d'un type de dossier, etc…).Si on doit modifier le processus achat par exemple, nous devrons modifier le code (ou le paramétrage) de cette application et s'assurer qu'il n'y a pas de régression (de bug suite à ce changement).

ET LES UTILISATEURS DANS TOUT ÇA ?

PCe courant marque le début de la « transformation digitale » des entreprises.

En mettant en place des Systèmes d'Information et des applicatifs, le métier structure ses processus et les outille. On commence ainsi à « engranger de la data ». À partir de là, les utilisateurs demandent des outils pour exploiter la donnée (Business Intelligence, Intelligence Opérationnelle, Analytics…).

La mise en place des applications de type monolithe, a permis d'adresser ainsi, de façon efficace, les besoins de transformation des métiers avec l'arrivée des outils et des « nouvelles technologies ». Cela a permis de mettre en place des réponses sur mesure pour des populations d'utilisateurs.

Alors, un monolithe c'est bien non ? Oui, mais nous avons voulu aller plus loin… et c'est ce que nous abordons dans la prochaine partie.

SOA .. ET LE MONOLITHE S'EFFRITA !

Les monolithes avaient tout pour réussir jusqu'à l'arrivée du SOA !

Dans la partie précédente, nous sommes revenus sur les tendances et enjeux métier qui avaient mené à la création d'applicatifs métier monolithique et à la mise en place des ERP.

Au début des années 2000, un changement s'opère. C'est le début du SOA (Service Oriented Architecture), mais aussi du Cloud.

ÉTAPE 2 : LE MONOLITHE VEUT COMMUNIQUER!

Lors de l'étape 1 nous avons vu que les applications métiers étaient faites d'un seul et unique bloc. Elles répondaient à un périmètre d'activité métier. Or, même avec la meilleure volonté du monde, il est très difficile de faire rentrer tous les processus de l'entreprise dans une seule et même application. Même avec un ERP puissant !

C'est ce que l'on retrouve dans les approches « best-of-breed » dans lesquelles, on choisit le meilleur outil possible pour une activité plutôt que d'avoir une seule solution pour tout faire quitte à ne pas bien répondre à 100 % des besoins. Elles ont l'avantage de mieux couvrir les attentes, mais imposent aussi certains choix d'architecture.

D'une façon ou d'une autre, certaines activités se retrouvent donc avec leur propre monolithe.

Pourtant, dans une entreprise, les différentes équipes travaillent toutes ensemble (c'est ce que l'ERP avait bien compris !). Une équipe Logistique a son propre système mais travaille avec la production ou les équipes commerciales ; une équipe Sinistre dans l'Assurance utilise des contrats, etc… De même, chaque monolithe n'est pas isolé et doit interagir avec les autres.
Alors, les applications se sont progressivement ouvertes.

La première ouverture a été de partager la donnée. On a donc intégré les applications entre elles. C'est l'essor des ETL (Extract Transform Load), des ESB (Enterprise Service Bus), des EAI (Enterprise Application Integration) et autres outils d'intégration.
Tout un outillage indispensable pour faire communiquer un Monolithe avec un autre.

LES LIMITES QUI ONT MENÉ AU SOA

Mais un gros problème de réactivité se pose alors par rapport aux besoins du business.

D'abord, parce que, d'une façon générale, ça prend du temps de faire communiquer les applications ensemble. Il faut comprendre les modèles techniques d'un côté et de l'autre, mais aussi l'utilisation fonctionnelle de la donnée. Ainsi, même avec des outils la mise en œuvre n'est pas immédiate ! Et une fois mis en place, le transfert d'information n'est pas toujours en temps réel. Beaucoup de ces transferts étaient liés à du batch, c'est-à-dire des traitements certes réguliers mais asynchrones (ex. une fois par jour).

De plus, cela crée des problèmes d'adhérence entre les systèmes. Si je modifie un applicatif (ex la structure technique ou fonctionnelle d'un objet « client »), il faut que je vérifie les impacts ailleurs. Ai-je supprimé une donnée utile ailleurs ? Des questions de gouvernance et d'architecture se posent alors.

Ainsi, si le business revoit sa façon de travailler ou met en œuvre un changement stratégique (ex. collaborer avec un partenaire, ou lancer un nouveau produit), il faut « un certain temps » pour réussir à mettre ça en place.

À partir de là, naît l'idée que, pour gagner du temps et s'orienter temps réel, les applications devraient pouvoir « communiquer directement entre elles ». C'est donc une idée assez étonnante énoncée comme cela, mais assez intéressant. Le SOA est né !

LE SOA C'EST QUOI?

Dans une « Service Oriented Architecture«, une application est « responsable » d'un périmètre de l'activité et doit fournir aux autres, les informations et fonctionnalités associés à ce périmètre.

Ainsi, une application de gestion de stocks, fournit à l'e-commerce, le niveau de stock d'un article sans recourir à une tuyauterie complexe pour transférer des données (qui ne seront peut-être pas suffisamment vite rafraîchies). Du coup, le site d'e-commerce appelle directement l'application de gestion stocks pour un article donné. L'application logistique lui renvoie le niveau de stock de l'article à la volée.

De même, les authentifications et droits d'accès sont désormais gérés dans une seule et même application. Ça évite d'aller les définir un peu partout dans le SI. Une application métier interroge alors le référentiel d'authentification qui lui répond en retour si l'utilisateur peut ou non se connecter.

Ces applications « exposent » des « services ».

C'est-à-dire qu'une fonctionnalité (une portion de leur code) peut être exécutée par une autre application. J'ai une Service S qui permet d'afficher les informations d'un client. Ce Service est accessible à un endroit et d'une façon prédéterminée. Pour appeler ce Service, l'autre application devra par exemple fournir le code du client (ID = 123) pour lequel je veux afficher les données. En retour, l'application qui appelle ce service recevra une réponse normée « Adresse : … ; Code Postal : … ; etc ».

Et à quoi ça sert de faire communiquer les applications entre elles ?

LES LIMITES QUI ONT MENÉ AU SOA

D'abord, l'architecture générale du SI est plus nette. Plus besoin de dupliquer la donnée ou même des fonctionnalités. Les interfaces « point-à-point » et les systèmes d'échange ne sont plus indispensables pour communiquer ! L'architecture d'ensemble donne la responsabilité (données + fonctionnalités) d'un périmètre fonctionnel à une application donnée.

Ensuite, en termes de déploiement et de mise en œuvre, on mutualise les fonctionnalités ! Si on lance une nouvelle application, elle n'aura pas besoin de prévoir les fonctions déjà gérées par d'autres applications.

Que ce soit des fonctions techniques (authentification, autorisations, …) ou métier (création, modification de données, de statut…) elle utilisera des fonctions existantes. Sa mise en œuvre en sera accélérée.

Alors certes des notions d'adhérence entre les applications et de gouvernance générale restent d'actualité. C'est normal et indispensable. Mais cette adhérence est beaucoup plus souple qu'avant. On définit seulement la façon dont les applications appellent une fonction et ce qu'elles reçoivent en retour. Elles sont complètement libres quant à l'utilisation qui est faite de l'information.

En termes de maintenabilité et d'évolutivité, c'est une grande avancée ! Je peux changer une application de fond en comble (techniquement ou fonctionnellement), du moment que les services exposés ne bougent pas, je n'impacte pas les autres applications !

Finalement, avec une architecture de type SOA, les premiers éléments de modularité du Système d'information font leur apparition. Les applications ne sont plus isolées et communiquent entre elles. Et qui plus en temps réel… à la volée… quand elles en ont besoin ! Je peux dès lors les associer un peu comme des briques de Lego…

Mais bien sûr, nous avons voulu aller plus loin.

L'évolution du SI continue !

DES SERVICES AUX MICROSERVICES

Nous avons précédemment évoqué les évolutions du SI qui nous ont fait passer des applicatifs monolithiques aux architectures SOA. La notion de service a amené à l'idée ensuite des microservices, qui sont une tendance de fond dans la conception actuelle des architectures IT.

LES LIMITES DU MONOLITHE

Pour comprendre l'idée des microservices, essayons de prendre un exemple opposé. Imaginons qu'il faille développer une application de gestion de devis. L'application devra par exemple comporter les fonctionnalités suivantes :

- je dois pouvoir me connecter (vérifier le mot de passe et autorisations)
- créer un devis
- enregistrer le devis
- générer un pdf de mon devis
- et l'envoyer par mail

L'application pourra ainsi être développée comme un seul et unique bloc. C'est la notion de monolithe.

En quoi est-ce problématique ?

Imaginons qu'il y ait un problème ou une évolution et qu'il faille modifier le code et redéployer l'application. Ou à l'inverse que tout fonctionne bien, mais que devant le succès de l'application et son utilisation massive, il faut augmenter les capacités allouées à la génération des pdf (par exemple). Il faudra donc modifier, déployer ou redimensionner toute l'application.

Cela va prendre du temps, arrêter l'ensemble de l'application et potentiellement redimensionner toute l'infrastructure. Il est donc risqué de modifier le monolithe et cela limite l'évolutivité du SI.

C'est là que les microservices interviennent.

ÉTAPE 3 : DES MICROSERVICES POUR DÉCOUPER LE MONOLITHE

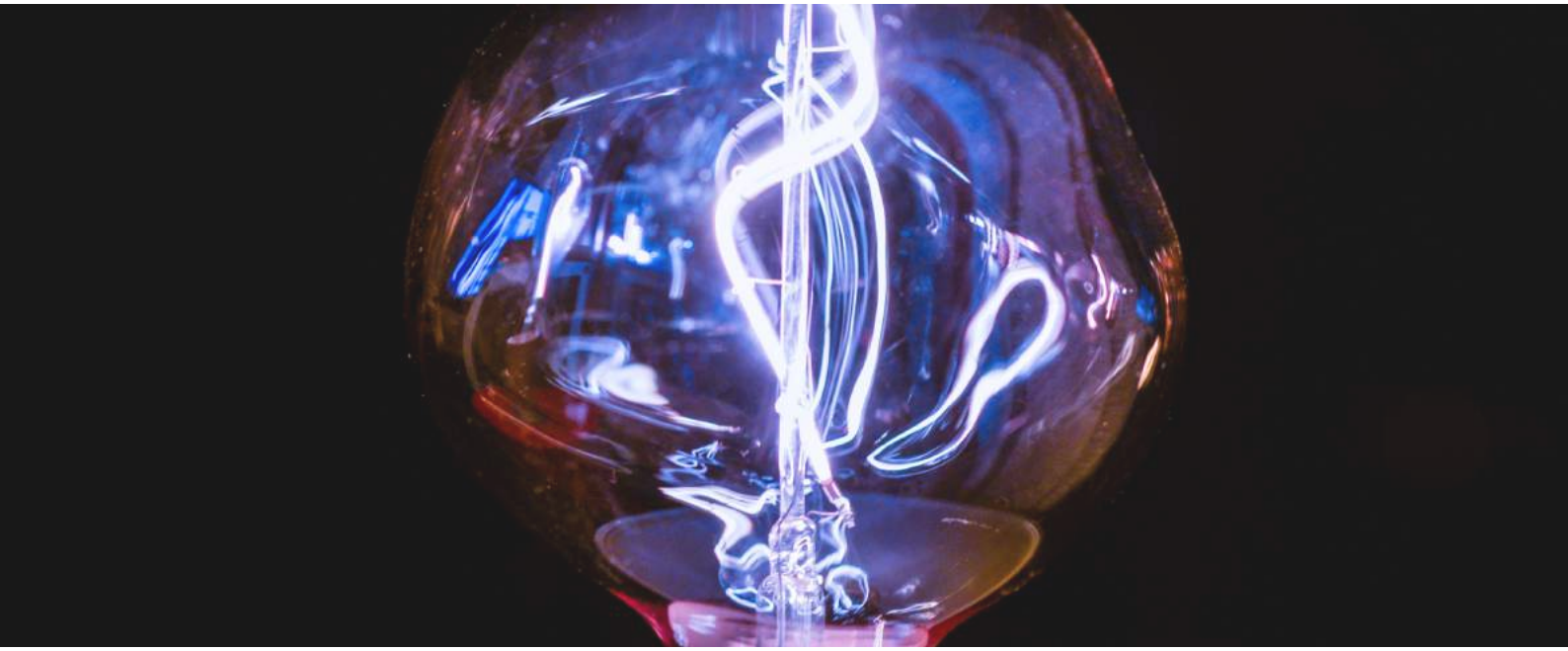
Ils sont la transposition du concept de service (SOA) entre les applications, au niveau de l'application métier elle-même.

Au lieu d'imaginer une application comme un ensemble de fonctions liées entre elles, chaque fonction est quasiment vue de façon autonome. La fonction est un service, complètement indépendant des autres.

Les microservices communiquent entre eux via des protocoles standardisés.

À l'extrême, on peut presque considérer que chaque microservice est une application à part entière. Reprenons l'exemple ci-dessus, les différentes lignes (connexion, création du devis, enregistrement…) sont autant de microservices différents. On va alors les concevoir et les faire vivre de façon complètement indépendante des autres.

Ainsi, nous pouvons alors faire évoluer les microservices, les déployer, les dimensionner beaucoup plus facilement et avec une grande souplesse.



LES AVANTAGES DES MICROSERVICES

IL Y A 4 AVANTAGES PRINCIPAUX AUX MICROSERVICES

D'abord, puisqu'ils apportent de la souplesse, ils donnent au métier, plus de réactivité par rapport à leurs orientations. En cas de changement, fonctionnel nous allons pouvoir faire évoluer chaque fonctionnalité de façon autonome. Et ainsi pouvoir « pousser en production » une mise à jour d'un microservice sans attendre les autres. Ainsi nous ferrons évoluer plus rapidement les fonctionnalités.

Ensuite, nous allons pouvoir dimensionner l'infrastructure de façon plus souple. Si un des services consomme plus de « ressources machine » nous pourrons lui en allouer rien qu'à lui et pas à toute l'application. Finalement, cela réduit les coûts d'infrastructure : à activité équivalente, je consomme moins avec les microservices !

ar ailleurs, l'architecture du code étant ainsi très modulaire, en cas de panne ou de bug, on va pouvoir détecter beaucoup plus facilement la source du problème.

On identifie directement le microservice défaillant et c'est celui-là qui est corrigé.

Enfin, chacun des microservices étant indépendant il va être développé dans son propre langage sans se soucier des autres. En termes d'organisation du travail et de performance globale de l'application c'est un plus énorme. On va pouvoir positionner des équipes différentes sur un même projet, chacun travaillant dans son propre langage. On peut aussi choisir les technologies les plus adaptées en fonction de ce qui doit être réalisé. De même, si une nouvelle technologie fait son apparition, on pourra la tester et l'intégrer facilement.

Rapidité, Réduction des coûts, réactivité, évolutivité et performance, les microservices ont tout pour plaire, non ? Voyons voir si on peut trouver encore mieux.

LE SERVERLESS FAIT ABSTRACTION DES SERVEURS

Dans une approche serverless, le développeur conçoit et met en œuvre une application et la déploie sur son environnement cloud. Le prestataire cloud gère alors de façon automatisée et dynamique les ressources qui vont être alloués aux diverses fonctionnalités et microservices.
Plus besoin de planifier, le prestataire garantit qu'il répondra à la demande.

ETAPE 4 : N'OUBLIEZ PAS LE CLOUD !

Puisque nous parlons d'évolutions dans l'IT, nous ne pouvons pas ne pas mentionner le Cloud.

Même si nous nous intéressons ici plus aux applications, qu'à l'infrastructure sous-jacente, le cloud change la donne.

La proposition de valeur du Cloud, c'est globalement de ne pas avoir à (trop) se soucier de l'infrastructure. Selon le bon vieil adage « loin des yeux, loin du cœur », les entreprises placent leur infrastructure IT dans le Cloud afin de moins s'en occuper et de faire des économies substantielles.

D'abord parce que l'infra est mutualisée entre plusieurs clients. Elle coûte marginalement moins cher que vous aviez avais la même quote-part d'infra chez vous. Ensuite parce qu'elle est plus souple. Si vous avez besoin de plus de puissance, le prestataire ré-alloue une partie de la capacité qu'il a déjà à sa disposition. Cela évite de commander à un fournisseur un serveur qu'il faudra installer. Avant, cela prenait au mieux des jours, au pire des mois d'avoir plus de puissance ; avec le cloud, cela devient quasi-instantané. Enfin, parce qu'une partie des tâches sont faites par le prestataire de façon automatique et plus économique (ex. installation, backup, ré-allocations).

Seulement, au niveau de la conception des applications, jusqu'à présent, le cloud n'a non plus bouleversé fondamentalement les choses. L'influence du cloud réside surtout dans le fait que les applications sont maintenant « conteneurisées » et qu'on les déploie sur des serveurs physiques ou virtuels, chez soi ou sur le cloud. Certes la notion de microservices a beaucoup de sens dans un contexte cloud, mais la promesse du cloud n'est pas pleinement atteinte pour l'instant : il faut encore s'occuper de l'infrastructure.

LE SERVERLESS, QU'EST-CE QUE C'EST ?

Ainsi, même pour des applications dans des conteneurs, même administrés avec des outils adéquats, il faut définir des ressources machine. Même dans le cloud, il faut prévoir et dimensionner un minimum ses environnements.

Pourtant, il est souvent compliqué d'imaginer à l'avance la puissance machine nécessaire. Le Business, même avec la meilleure volonté du monde a du mal à prévoir les pics de fin d'année, les Black Fridays, les promotions de la concurrence, le succès d'une nouvelle activité ou d'une promotion.

Le résultat, c'est que même dans le cloud, on peut avoir des pannes ou un manque de ressource machine. Même dans le cloud, on peut avoir des ressources non exploitées (et pourtant facturées !).

À partir de là, apparaît la notion de « serverless ». L'idée du serverless, contrairement à ce que le terme pourrait laisser entendre n'est pas de supprimer un serveur. L'idée est tout simplement de laisser le prestataire cloud gérer à 100 % les ressources du serveur. Et dans ce cas-là, la promesse du cloud serait véritablement atteinte : plus besoin de s'occuper de l'infrastructure !

Dans une approche serverless, le développeur conçoit et met en œuvre une application et la déploie sur son environnement cloud. Le prestataire cloud gère alors de façon automatisée et dynamique les ressources qui vont être alloués aux diverses fonctionnalités et microservices. Plus besoin de planifier, le prestataire garantit qu'il répondra à la demande.

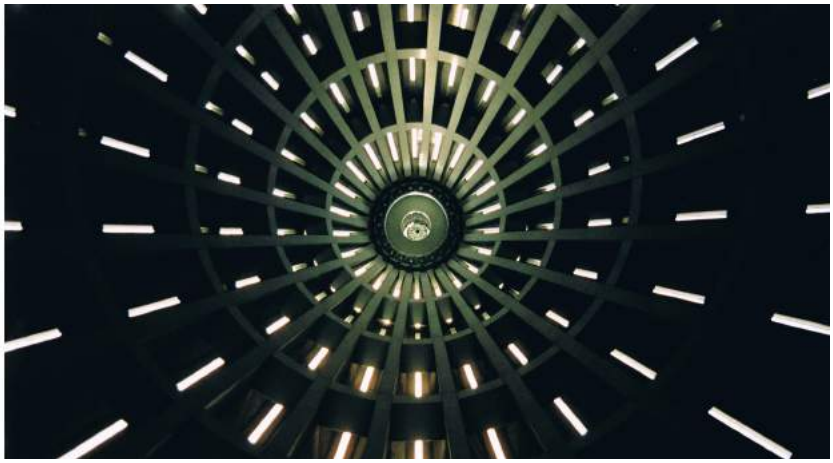
LE SERVERLESS A DE NOMBREUX AVANTAGES

On va donc comme toujours pouvoir déployer plus vite. On allait déjà plus vite en découpant l'application en microservices. Maintenant, vu que je m'occupe moins des préoccupations d'infrastructure, donc je vais plus vite.
L'autre avantage évident est qu'on a normalement une meilleure performance et une plus grande disponibilité. Et pour cause, c'est le prestataire qui s'en occupe et moi, je n'en porte plus la responsabilité !

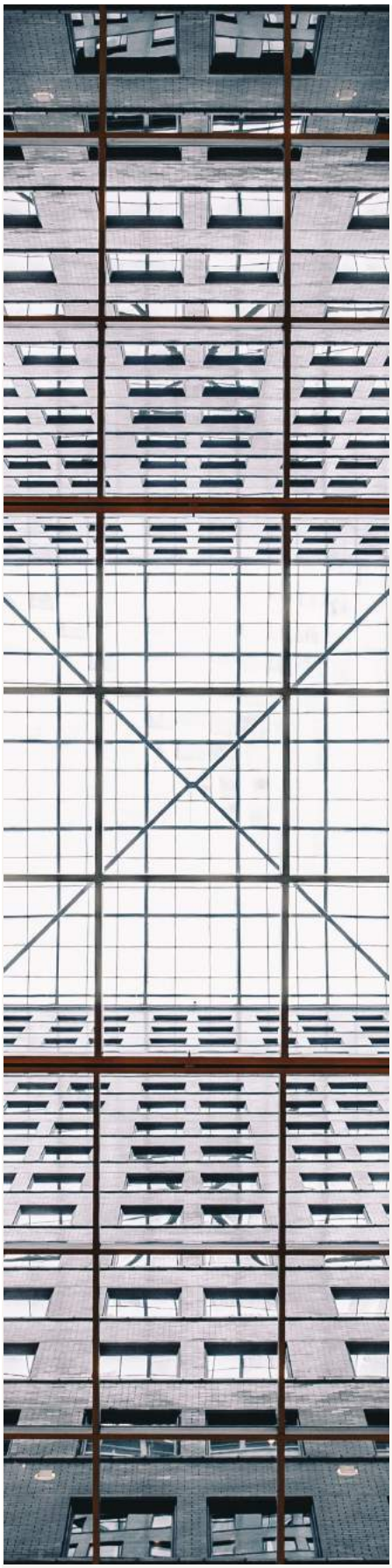
Par ailleurs, cela joue sur les coûts. Puisque les ressources allouées sont toujours en ligne avec l'utilisation, il ne faut surtout pas payer de ressource que l'on n'utilise pas ! Cela présente aussi un intérêt environnemental, puisqu'on a théoriquement moins de gâchis. L'efficience est aujourd'hui une priorité. On manque toutefois encore trop de recul sur le serverless pour évaluer sa performance écologique.

Enfin, toujours sur les coûts, les fournisseurs de Serverless (ex. Firebase de Google, Amazon et Microsoft) facturent à l'usage… Là encore, on ne paye que ce qu'on consomme. On variabilise donc un coût qui serait fixe en temps normal (le coût d'un serveur acheté ou loué).

Globalement l'approche serverless s'inscrit dans une démarche continue de modularité et de souplesse de l'IT.



… ET QUELQUES INCONVÉNIENTS !



La tendance du serverless est encore trop récente pour n'avoir que des avantages !

Les coûts variables, c'est bien, mais il faut les surveiller pour éviter qu'ils ne s'envolent. En tout cas, cela peut amener à revoir son pricing model. Il ne faut surtout pas que le métier se retrouve avec un prix de vente fixe et des coûts qui, eux, explosent parce qu'ils sont variables !

Une autre limite actuelle au serverless, c'est que pour profiter pleinement de l'infrastructure cloud, il faut tout de même adapter son code à la plateforme Cloud cible et utiliser des fonctions mises à disposition par le prestataire. Autrement dit, votre code Serverless pour Amazon sera légèrement différent de celui pour Microsoft ou Google. Cela créé donc une adhérence et limitera peut-être la possibilité de changer de prestataire dans la durée. Bien sûr des petits malins viendront proposer une couche d'abstraction qui permettra d'être agnostique par rapport au Cloud cible. Les premiers frameworks ont fait leur apparition.

Enfin, dernière précision. On présente parfois le serverless comme la fin du DevOps. C'est une erreur. En effet, les mécaniques d'intégration et de déploiement continus restent d'actualité. Et il faut que pendant le DEV on intègre les contraintes OPS (cf. le point sur l'adhérence ci-dessus). Ça ne rend donc pas le DevOps obsolète. Au mieux, ce que cela simplifie, c'est la partie OPS. Mais, c'est déjà beaucoup !
En tout cas, c'est la promesse du serverless… qu'il faudra suivre de près dans les prochaines années. Il faudra notamment voir quels sont les langages et les outils les plus adaptés à cette nouvelle tendance !

Blog

mind7
CONSULTING