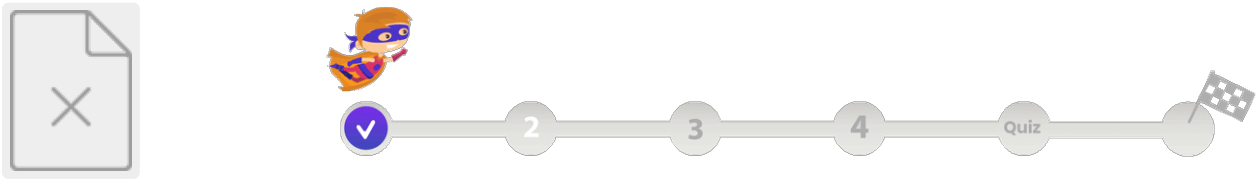
2 - L'Architecture MicroServices - MSA



2. L'architecture MicroServices - MSA

2.1 Les Microservices c’est quoi ?

> Selon Wikipédia, "**Microservices** is a variant of the service-oriented architecture (SOA) architectural style that structures an application as a collection of loosely coupled services. In a microservices architecture, services are fine-grained and the protocols are lightweight. The benefit of decomposing an application into different smaller services is that it improves modularity. It also parallelizes development by enabling small autonomous teams to develop, deploy and scale their respective services independently. It also allows the architecture of an individual service to emerge through continuous refactoring. Microservices-based architectures enable continuous delivery and deployment.

L'architecture en microservices (MSA) désigne donc une **conception logicielle basée sur l'architecture orientée service** (SOA) en opposition aux architectures dites monolithiques (qui sont faites d'une seule pierre).
La particularité d'un microservice, comme son nom l'indique, provient de la **structure** des services. Ils doivent être de **petite taille** et utiliser u**n protocole de communication très léger**.

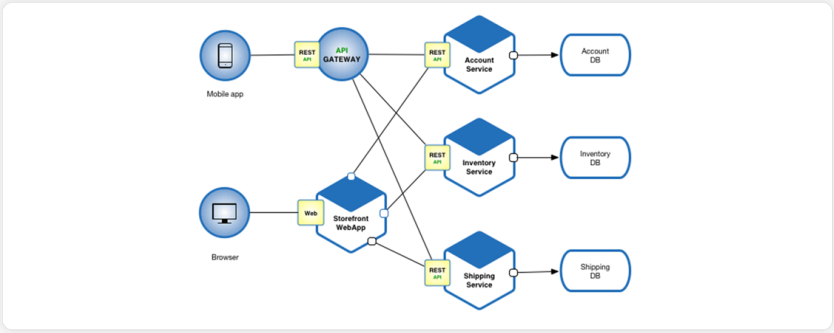


Figure 1 : Exemple d'architecture microservices

2.2 Qui les utilise et pourquoi ?

Les sites les plus en vogue du moment tels que **Netflix, Amazon, Uber, SoundCloud ou Google** ont fait le pari de migrer vers **une architecture en microservice** il y a plusieurs années. Inutile de vanter la réussite de ces entreprises aujourd’hui, elles avaient pourtant des défis de taille à relever. Leurs projets informatiques étaient titanesques. Tout projet informatique à tendance à grossir avec le temps, c’est un problème auquel les grandes entreprises se trouvent très vite confrontés, des fonctionnalités sont ajoutées petit à petit, les obsolètes sont rarement supprimées.

2.3 Les problématiques des gros projets

Comme vu précédemment, maintenir de gros projets dans le temps peut s'avérer douloureux.

a. Les problèmes de complexité

On trouve trois désavantages majeurs issue d'un problème de complexité :

*** L'évolutivité**

Plus un projet avance, plus les interactions entre les différentes briques deviennent complexes. Malgré le découpage logiciel en composants, il y aura toujours des patchs sauvages qui seront appliqué à la va vite et qui réduiront la lisibilité du code et la capacité à apporter de nouvelles fonctionnalités.

*** La fiabilité**

Le problème d'évolutivité décrit ci-dessus entraîne une augmentation des effets de bords rendant encore plus compliqué les remaniements de code.

*** La scalabilité**

Plus un projet prend de l'ampleur, plus il devient **compliqué, risqué et couteux** de modifier sa structure principale : c'est pourtant systématiquement nécessaire lorsqu'il faut améliorer sa scalabilité, sa capacité à être déployé de façon multiple pour tenir une charge applicative plus grande.

b. Les problèmes d'innovation

De la même façon que pour les problèmes de complexité, **le temps va devenir un facteur bloquant** en terme d'innovation aussi bien d'un point de vue technologique que métier.

*** Innovation technologique**

Idéalement, afin de capitaliser les technologies et les savoirs, il est primordial pour une entreprise d'utiliser les **mêmes langages** et les **mêmes outils pour tous les projets informatiques.** Malheureusement, il est rare qu'un langage soit à même de répondre à toutes les problématiques que peut rencontrer une société. Quand on parle de grand projet, le constat est le même. Il faut essayer de propager toutes les évolutions techniques au sein d'un seul et unique projet, ce qui devient un véritable challenge avec le temps.

*** Innovation métier**

Afin de rester dans un secteur de pointe et ne pas se faire dépasser par la concurrence, il faut quotidiennement répondre aux besoins métier. Pour cela il faut aménager les projets existants en conséquence. Malheureusement, plus un projet est gros, moins il sera facile de le modifier afin d'essayer de nouveaux produits ou d'attaquer de nouveaux marchés.

2.4 Les avantages des MSA

Nous allons voir comment une approche MSA permet de résoudre les problèmes de complexité et d'innovation que nous avons identifié précédemment sur les gros projets.

a. Les problèmes de complexité

On retrouve nos trois complications issues des problèmes de complexité :

*** L'évolutivité**

On oublie ici le concept de dette technique. Sur des petits projets, il n'y a plus de problème d'évolutivité. L'architecture étant moins complexe toute l'équipe en charge du projet connaît les moindres recoins du code. Il est donc plus facile d'ajouter une fonctionnalité à un microservice, voire de créer un nouveau service dédié à cette fonctionnalité.

*** La fiabilité**

La communication entre chaque service est codifiée et unifiée, que ce soit en passant par des appels vers des API ou via des systèmes de message. La gestion des erreurs est donc déportée sur chaque service ce qui permet d'avoir une granularité plus fine pour la résolution de problème.

*** La scalabilité**

En se basant sur des petites fonctionnalités, il est beaucoup moins coûteux de remanier du code voire même de repartir du début lorsque l'on rencontre des problèmes de scalabilité qui impactent un service.

b. Les problèmes d'innovation

De la même façon que pour les problèmes de complexité, l'approche MSA va nous permettre de regagner en innovation.

*** Innovation technologique**

Chaque équipe peut faire les choix technologiques qu'elle considère les plus adaptés à ses services puisqu'elle n'impacte plus les autres équipes.

*** Innovation métier**

Maintenant que tous les projets sont découpés en microservices, il devient moins coûteux et compliqué de lancer un nouveau projet innovant en capitalisant sur les projets déjà existants. Il y a également plus de chances de pouvoir réutiliser du code existant.