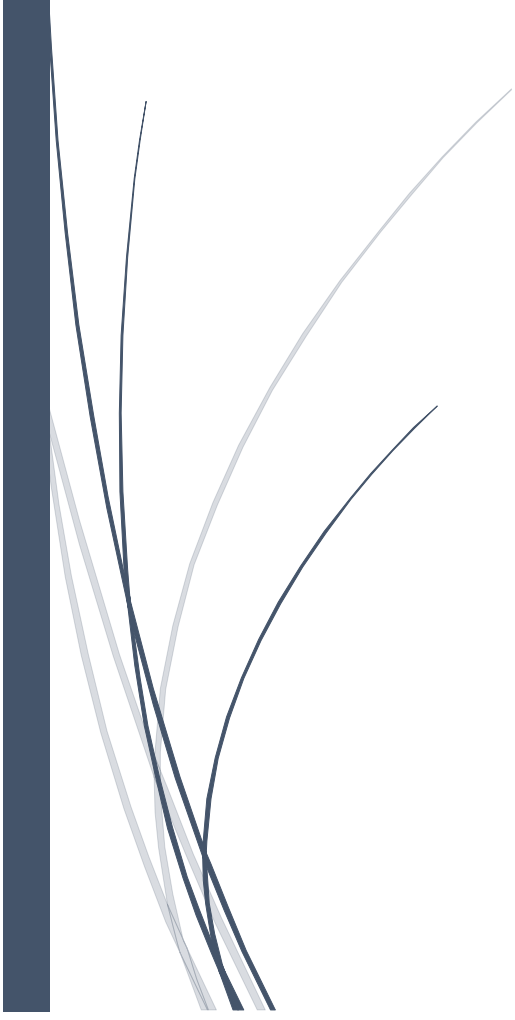
14/06/2017

Architecture MicroServices

Génie Logiciel, Sciences-U



Jérôme BADIN ; Christopher SEBASTIAO

Table des matières

I - Les Microservices, c'est quoi? (Contexte, définition, caractéristiques)

Les architectures micro-services permettent de développer, de déployer et de gérer opérationnellement des applications distribuées, constituées de services aux fonctionnalités complémentaires, potentiellement hétérogènes et interopérables.

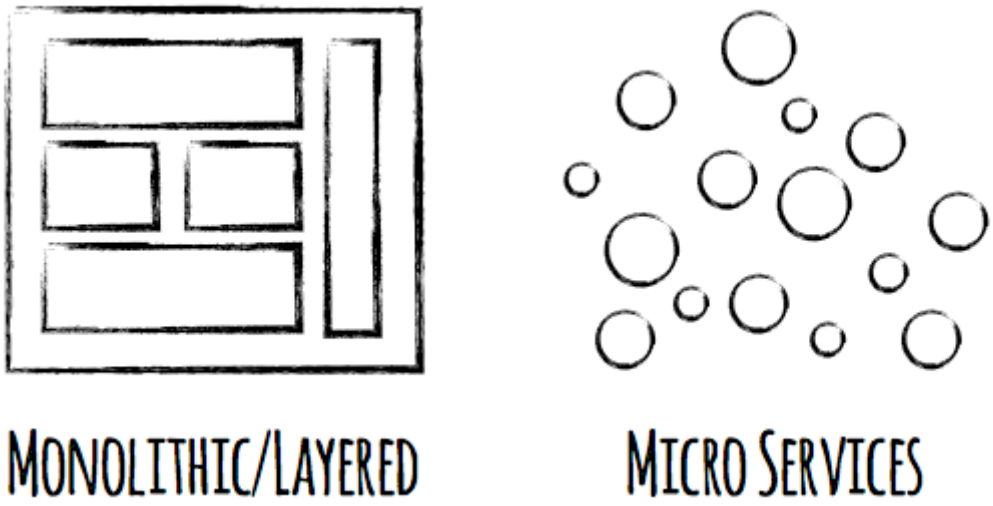
Ce sont pour les systèmes complexes que les microservices sont les plus bénéfiques

II – Réponse à quelle problématique ?

Après plusieurs années de développement logiciel et de maintenance, certaines applications d'entreprise s'avèrent laborieuses et trop coûteuses à faire évoluer. Ce type de dette technologique est un constat, une difficulté majeure, que rencontrent à terme de nombreuses entreprises.

Cela conduit souvent à faire table rase des développements passés, et à les reconstruire à partir de zéro. L'essor des **architectures micro-services** (micro-services architectures – *MSA*) répond à cette problématique, et propose des moyens de la résoudre.

III – Architecture Monolithique VS architecture Microservices

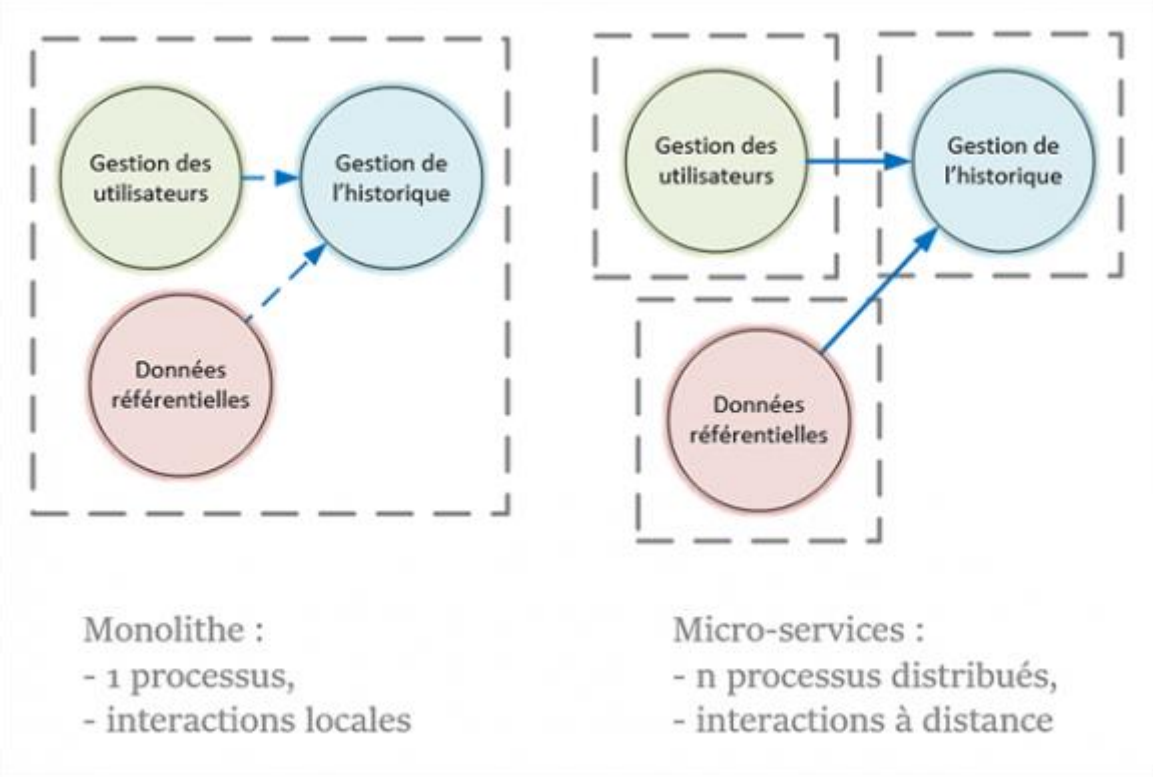


Une application monolithe est une application constituée d'un seul bloc, dont l'ambition est de traiter toutes les demandes que nous leur soumettons. Ces applications sont écrites dans un langage unique. Et à chaque ajout elles deviennent de plus en plus complexes, difficiles à maintenir et déployer.

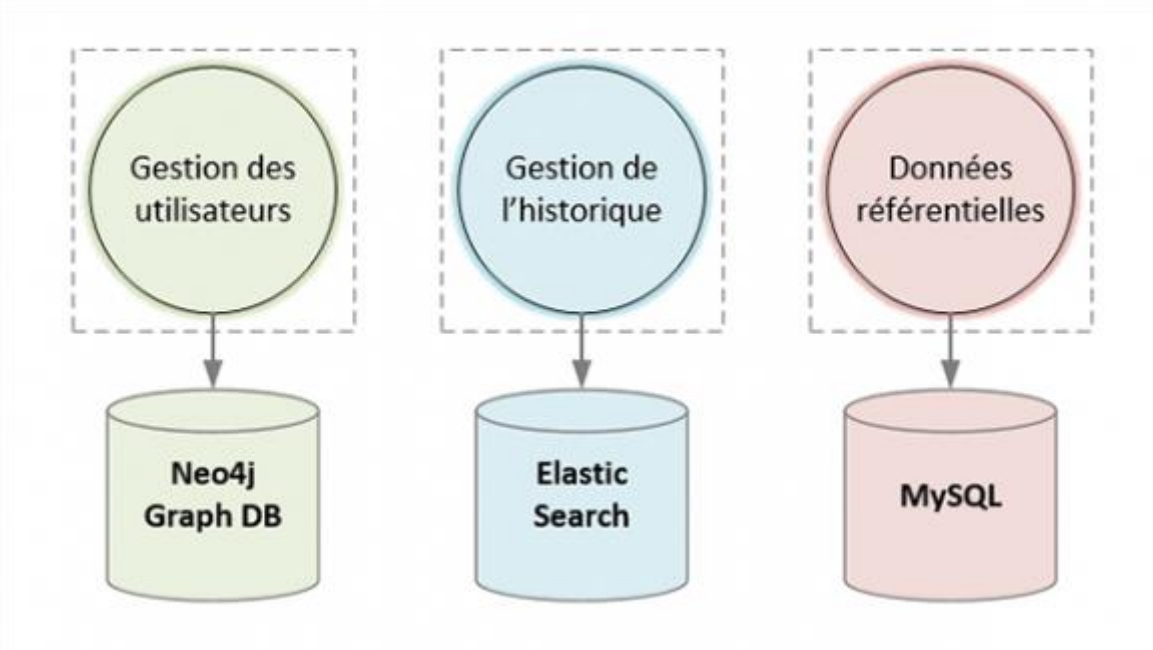
Les Monolithes obligent les entreprises à répartir leurs ressources par projet. Ainsi on empile autant de consultants et de développeurs qu'il existe de projets. Une multitude de strates s'interposent entre le développeur et l'utilisateur final.

Le Monolithe devient une fin en soi. C'est autour de lui que s'organisent les équipes, que se concentre l'énergie.

Et c'est là que les Micro Services deviennent révolutionnaires. Ils permettent d'organiser autour des Monolithes une série de petits services autonomes, connectés entre eux par des APIs.



Les Microservices étant indépendants et interopérables, ils ne sont pas contraints à une uniformité technologique. Les choix technologiques peuvent ainsi être adaptés aux besoins spécifiques de chaque micro-service. Il est ainsi possible d'utiliser un serveur de bases de données non-relationnel plus performant en écriture et plus extensible pour un micro-service de gestion d'historique.



IV - Les avantage & inconvénient des Microservices

Avantages :

- Déploiements et retombées plus rapides et plus simples
- Contrôle modulaire
- Facilite le travail en équipe
- Augmente la qualité globale des applications.
- Permet d'utiliser le meilleur langage de programmation en fonction de la problématique.
- Permet d'automatiser les tests, la qualification et le déploiement en production.
- Chaque Microservices peut être mis à l'échelle indépendamment et peut être optimisé sans affecter le reste de l'application.
- Augmentation de la robustesse de l'architecture, de la tolérance aux pannes
- Vitesse de livraison indépendante (par différentes équipes)
- Cadre approprié / outil / langue pour chaque domaine
- Composant de recommandation à l'aide de Python ?, Service de catalogue en Java ...
- Plus grande résilience
- De défaut d'isolement
- Meilleure disponibilité
- Agilité
    - Conception à l'échelle de la fonctionnalité
    - Isolation des fonctionnalités
- Légèreté
    - Permet de s'abstraire de la couche OS (VM)
- Scalabilité
    - Verticale : permet de ne répliquer que les services chargés
    - Horizontale : déport des services les plus chargés vers des nœuds différents

Inconvénients :

- Overhead
    - Nécessite une communication orientée-message entre les services (vs. appel de méthodes)
    - Nécessite une « surveillance » du fonctionnement des services (monitoring, tolérance aux pannes, pilotage)
    - Accès aux ressources partagées
    - Gestion décentralisée des données
    - Application polyglotte.
    - Augmentation du trafic réseau.
    - Coût initial plus élevé.
    - Sécurité des communications entre les Applis.
- Humain
    - Nécessite que les équipes soient effectivement structurées selon l'architecture du produit et necessite un bon niveau d'expertise DevOps
- Vision / mise au point globale de l'application
    - Pas triviale, peut nécessiter plusieurs itérations

V - Pièges d'une utilisation abusive des Microservices

- Granularité
  - J'ai regroupé plusieurs services en un seul « macroservice »
  - Chaque méthode de mon application est devenue un microservice
- Architecture globale
  - Il est très difficile de comprendre ce qui se passe entre mes services
  - La recherche d'information dans les logs est trop longue
  - Impossible de relancer ou ajouter rapidement des instances de mes services

VI - Pourquoi migrer vers les Microservices ?

- Frustration de ne pas obtenir le résultat souhaité avec une architecture monolithique.
- Arrivée sur le marché d'outils permettant le déploiement des applications Microservices avec plus de facilité.
- Large adoption des solutions d'infrastructure en tant que service (IaaS).
- Le passage des grosses sociétés du Web vers des architectures complètement Microservices.

VII – Vers la fin des applications monolithiques ?

Les microservices ne constituent pas non plus une recette universelle. Un système peu complexe sans problématique de charge ou dont le rythme d'évolution est faible ne bénéficiera pas de l'approche microservices.

VIII – Synthèse rapide

Architecture

Application monolithiques :	Application Microservices :
• Applications N-tiers. • IHM, core, accès aux BDD. • Chaque modification nécessite de redéployer la • totalité de l'application. • Chaque modification nécessite de retester l'ensemble • de l'application. • Difficile de garder au fil du temps une bonne structure • modulaire. • Mise à l'echelle couteuse. • Manque de diversité technologique. • Pas facile de changer un composant	• ÉLASTIQUE Un Microservice doit pouvoir être déployé un nombre de fois qui varie en fonction de la demande, et ce, indépendamment des autres services dans la même application. • RÉSILENT Un Microservice doit échouer sans affecter d'autres services dans la même application. • API Les Microservices doivent avoir une API stable, cohérente et bien documentée. https://openapis.org/ • MINIMAL MAIS COMPLET Un Microservice doit être le plus petit possible mais pas plus petit. Il doit offrir une fonction complète avec des dépendances minimales avec les autres services.

Pizza-Teams

Pour éviter les problèmes des gros projets, il suffit de n'avoir que des petits projets.

On va donc limiter la taille des projets à quelques personnes pour avoir des pizza teams d'une taille maximale de huit personnes tout compris afin que l'équipe soit créative, mais pas au point que la cohésion et la communication se perdent.

Il ne s'agit pas de séparer les gros projets en sous-équipes mais bien de projets *indépendants* : chacun a son organisation, son calendrier, sa base de code et ses données.

Cette technique permet également d'ajouter de la qualité à l'application. Qualité temps au niveau humain qu'au niveau mircroservices