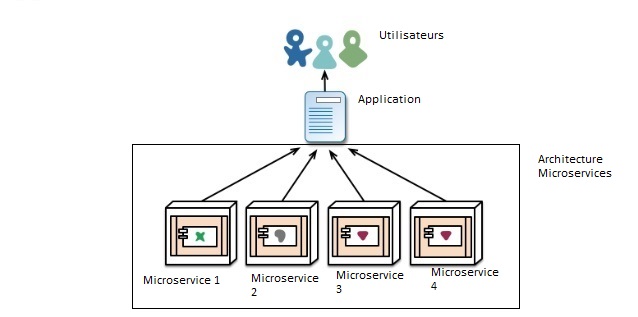
Architecture Microservices

***I)L’architecture Microservices***

*1)que ce que l’architecture Microservices?*

Il s’agit avant tout d’une méthode de développement logicielle, se basant sur l’idée que chaque fonctionnalités de l’application est un service effectué par une unité fonctionnelle, indépendant des autres unités et modulaires. On peut représenter une telle architecture ainsi:



Ces microservices correspondent donc à une fonctionnalitée précise,logique et en adéquation avec le système. Chacun d’entre eux possède son propre code, gère et manipule ses propres données et ne les partages pas directement avec les autres microservices.

Les communications entre eux sont effectuées par un intermédiaire, comme une API REST ou un bus de messages. Cette communication fonctionne à l’aide de mécanisme et de messages légers, simple et standardisés . De ce fait, chaque microservices peuvent être créé avec des technologies différentes: Là où un est codé en C# avec une base de données type Mysql, un autre peut être fait avec du Ruby et possède une base de données NoSql.

Les microservices possède une haute tolérance au pannes. De leur nature indépendante, si un microservice est sujet à une panne, l’ensemble de l’application continue de fonctionner.

Finalement, ils sont hautement modulables. En cas de pannes ou de modification d’un microservice, il suffit simplement de le “dépluger” de l’architecture, modifier ou corriger ce que l’on souhaite et redéployer le microservice. Durant le temps de la modification, l’ensemble de l’application continue à fonctionner et il n’y a pas nécessité de redéployer l’ensemble de l’architecture.

*2) la “Componentization” d’un microservices.*

Chaque microservices correspond à un service. C’est ce qu'on appelle la “Componentization”. Cela se distingue par l’utilisation de librairies réutilisables afin de pouvoir déployer les microservices de manière indépendantes.

Cette Componentization répond à des 4 principes:

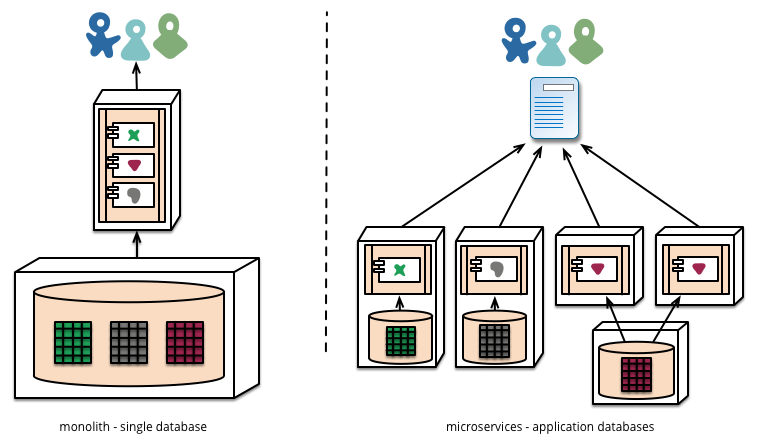
-Cohérence et logique du système: Tout service doit correspondre à une fonctionnalité précise.

-Spécificités des fonctionnalités: Un service ne fait que une chose, mais il l’a fait bien. (Cela peut rappeler le principe Kiss, ou Keep it Simple and Stupid, où un serveur, par exemple, fait une chose simple,et le fait bien)

-autonomie des services: Chaque service possède ses données et ne les partages pas (ou alors indirectement) avec un autre.

-Indépendance des microservices: Couplage faible avec un autre microservice.

***II) Opposition entre Monolithique et Microservices***



Ici, nous avons une application en monolithique, dont les services utilisent les mêmes bases de données: Ils gérent et manipulent les mêmes données.

Dans une architecture Microservices, on peut tomber sur deux cas différents, de manière générale:

1)Chaque microservices possèdent sa propre base de données.

2)deux microservices sont redondées. Dans ce cas, ils utilisent les mêmes données, donc la même base de données.

***III)Les deux types principaux de communications***

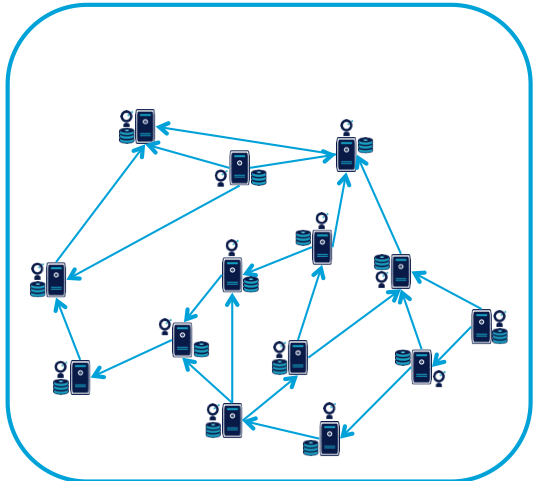
Les microservices s’échangent les informations, sous forme de messages simple et standards, via un intermédiaire. Il existe deux types d'intermédiaires principalement utilisé: L’API REST ou le bus de messages.

1. *API REST*

a)définition

REST est l’acronyme de “Representational Statie Transfer”. Il utilise des standards comme les URI, le protocole HTTP ou HTTPS, ou le type MIME (text/xml,text/html,etc…).

Dans un tel mode de fonctionnement, chaque services possédera une interface ayant une adresse IP, ainsi que la liste des adresses de chacun des services. Cela crée des “liens” entre chaque services:



b) avantages

-Chaque services à une base de code plus petite, ce qui la rend plus lisible et facilite sa maintenance.

-Il s’agit de la méthode la plus simple à mettre en place et la préférée quand il s’agit de migrer depuis une application monolithique.

-le protocole utilisé, HTTP/S, est standardisé, et possède une très haute interopérabilité.

-Efficace quand le nombre de microservices est réduit.

c) désavantages

-L’API REST se base sur les adresses IP des microservices pour connaître la destination du messages. Si une de ces adresses change, il faut changer l’intégralité des configuration utilisant ce service.

-Gérer les multiples instances d’un service avec une configuration par adresse IP est très contraignant. Cela oblige à maintenir manuellement l’inventaire des services utilisés.

-Transposer une application monolithique en microservices en remplaçant un appel de méthode par son équivalent en Rest limite les performances globales du service.

-Peut devenir une faiblesse quand le nombre de services se multiplie. Au delà de la complexité de gestion de l’ensemble des adresses IP, on peut arriver à une situation où le nombre de requêtes entre chaque services deviennent trop important, polluant le réseau.

*2) Bus de messages*

a) définition

Un bus de messages est un élément d’infrastructure qui fait l’intermédiaire entre les différents services. Son comportement est proche d’un switch ou d’un ESB. Chaque microservices vont pointer sur ce bus, qui va récupérer les messages pour les distribuer aux services destinataires.

Chaque service ne connaît donc que le bus de messages comme intermédiaire, et celui ci n’a comme unique responsabilité de transmettre les messages. Il est donc “sans intelligence”. C’est à dire que il n’effectue pas de traitement de données.

b) avantages

-Les microservices sont réellement isolés du reste du système. Leur seuls liens vers l’extérieur étant la typo des messages (si le message à été émis ou reçus), ainsi que le bus.

-Le système est asynchrone, évitant des lenteurs. Une fois le message envoyé, le service n’attend pas de réponse immédiate. Si le destinataire devient lent, l’appelant n’est pas impacté.

-Améliore la tolérance à la panne. Si un service tombe, tous les messages qui lui sont destinés sont gardés dans le bus. Il pourra reprendre le travail une fois redémarré, sans perte.

-Evite la pollution du réseau par un trop grand nombre de requêtes ou de messages. Tous les messages vont vers le Bus, qui les renvoi. Cela centralise les échanges, évitant une pollution sur l’ensemble du réseau et donc des effets de ralentissements.

c) désavantages

-De ce nature, le bus de messages devient un SPOF (single point of failure). Si il est soumis à une panne, les communications entre services ne fonctionne plus.

-Plus complexe à mettre en place et configurer que une API REST.

-Son asynchronisme est autant un point fort que faible. Un service peut tomber sans que cela soit visible immédiatement. Les autres services continueront d’envoyer leur messages sans qu'un ralentissement soit apparent, jusqu’au moment où on a besoin du service qui est en panne. Il faut donc réfléchir à un bon système de monitoring.

-La sécurité du bus de messages. Celui ci n’étant qu'un élément intermédiaire entre les différents services, la sécurité native du bus est quasi inexistante. Du fait que tous les messages transitent par celui ci, un minimum de sécurité doit être mis en place.

***IV) L’impact de l’architecture microservice dans les équipe de développement***

L’architecture microservices à importée une nouvelle manière de penser pour les équipes de développement. En miroir des services, elles s’organisent autour de fonctionnalités métiers et non autour de technologies. Les équipes deviennent alors multidisciplinaires et se complètent pour la réalisation d’un objectif: faire fonctionner le système.

A chaque service va donc s’organiser une équipe, souvent constitué de 5 à 7 personnes, qui va s'approprier le service, l’accompagnant durant toutes les étapes de sa vie. Le service devient donc un produit de l’équipe.

Ces équipes auront l’opportunité de choisir les technologies et outils les plus adaptés selon eux à remplir l’objectif, et sauront pourquoi il développe le service, facteur pouvant être important dans le succès de sa mise en place.

***V) Microservices, Avantages et inconvénients***

*1)avantages*

-Développement et déploiement indépendant. On peut facilement prototyper, déployer et tester une nouvelle fonctionnalité sans que cela est d’incidence avec le reste de l’architecture.

-développer par une petite équipe.

-souplesse en terme de technologie utilisée. Les équipes choisissent les technologies utilisés.

-Isolation des pannes.

-Le contenu des messages n’est pas pollué par une multitude d’enveloppes techniques. Ils n’ont que les données pertinentes.

-la granularité des microservices permets un meilleur contrôle du cycle de vie d’une fonction.

*2)désavantages*

-Mise en place de système de monitoring. A cause de cette grande tolérance à la panne, une panne ne sera pas remarquée avant un certains moment si on ne surveille pas l’architecture, pouvant amener à des situations problématiques.

-Peut être complexe de comprendre le système dans sa globalité. Comprendre un service en sois n’est pas dur. Mais comprendre sa place dans l’ensemble du système peut être complexe.

-La granularité du découpage des microservices. Soit on à trop de microservices, à force de découper les services et fonctionnalités, arrivant à une architecture complexe à comprendre et surveiller, rendant les performances mauvaises. Soit on se retrouve avec un microservice possédant plusieurs fonctionnalités qui se ressemble. Dans ce cas, dès que le comportement d’une de ses fonctionnalités se met à évoluer et changer radicalement, elle doit devenir un microservice.

-La gestion des logs est complexe, étant donnés que l’architecture est éclatés entre différents services. Cela rend compliqués le débogage et l’analyse d’un problème en profondeur.