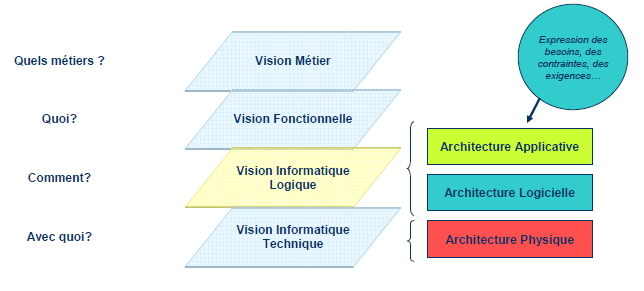
Architectures Applicatives

**Définition :**

Une architecture d'applications décrit le comportement des applications utilisées dans une entreprise, en mettant l'accent sur la façon dont elles interagissent les unes avec les autres et avec les utilisateurs. Il est axé sur les données consommées et produites par les applications plutôt que sur leur structure interne.

L'architecture des applications signifie gérer la manière dont plusieurs applications sont susceptibles de fonctionner ensemble. Il est différent de l'architecture logicielle, qui traite de la conception technique de la construction d'un système.



|  |  |
| --- | --- |
| Architecture Applicative | Architecture logicielle |
| Blocs fonctionnels,  Référentiels de données,  Flux de donnés | Module logiciel,  Composant,  Classe |
| Des applications | Une application |
| Processus métier, activités | Spécifications fonctionnelles |
| Spécifications techniques de l’ensemble du S.I  (ensemble des composants et modèles de données | Spécification des classes logicielles |

L'architecture des applications est spécifiée sur la base des besoins métier et fonctionnels. Cela implique de définir l'interaction entre les packages d'applications, les bases de données et les systèmes middleware en termes de couverture fonctionnelle. Cela aide à identifier les problèmes d'intégration ou les lacunes dans la couverture fonctionnelle.

L'architecture des applications tente de garantir que la suite d'applications utilisée par une organisation pour créer l'architecture composite est évolutive, fiable, disponible et gérable.

Aucune méthodologie n'a réussi à s'imposer pour la conception d'architecture.

Les différentes architectures applicatives :

1. Architecture monolithique

Une architecture monolithique est le modèle unifié traditionnel, composé en une seule pièce pour la conception d'un logiciel.

Le logiciel monolithique est conçu pour être autonome, les composants du programme sont interconnectés et interdépendants. C’est une architecture étroitement couplée, chaque composant et ses composants associés doivent être présents pour que le code soit exécuté ou compilé.

Au début d’un projet, quand celui-ci est encore de petite taille, elle reste très simple à comprendre favorisant la maintenabilité et la testabilité. Le déploiement, facile et rapide à mettre en place. Cependant, au cours de l'évolution d’un projet, cette architecture montre fréquemment ses limites.

Si en début de projet, la simplicité du modèle monolithique favorise grandement la compréhension du code, ceci devient de moins en moins vrai au fur et à mesure que le volume de code augmente, et ce, même avec une organisation rigoureuse de ce dernier.

**« Inconvénients » :**

Les applications monolithiques sont réalisées dans une seule technologie alors qu’il est intéressant d’exploiter les capacités de différents langages pour des buts spécifiques. Cela a pour effet pervers de rendre une application dépendante à une technologie et à ses compatibilités.

Une modification apportée au sein d’un élément peut créer des changements imprévus au sein d’autres éléments.

De plus, si n'importe quel composant de programme doit être mis à jour, l'application entière doit être réécrite, alors que dans une application modulaire, n'importe quel module séparé (tel qu'un microservice) peut être changé sans affecter d'autres parties du programme.

**« Avantage » :**

Cependant, dans de nombreux cas d'utilisation, les architectures monolithiques présentent des avantages non négligeables et durables que nous ne pouvons pas simplement négliger parce qu'elles n'adhèrent pas à un modèle moderne. Les architectures microservices peuvent introduire une complexité significative qui n'est pas toujours nécessaire.

Les programmes monolithiques ont généralement un meilleur débit et de meilleures performances pour les applications qui ne sont pas destinées à devenir trop grandes car ils n’ont pas à envoyer plusieurs appels d’API à plusieurs microservices différents.

1. SOA (Service Oriented Architecture)

Compte tenu des difficultés et inconvénients énoncés plus haut, de nouvelles méthodes ont vu le jour et se sont largement répandues au sein de grandes entreprises intervenants dans le domaine du web.

SOA (Service Oriented Architecture) pour Architecture Orientée Service est apparue dans les années 2003/2004. C’est une architecture logicielle qui s’appuie sur différents services simples. Elle permet de subdiviser une fonctionnalité en un ensemble de fonctions concrètes et basiques, nommé service. Cette architecture est plus qu’une nouvelle technologie car elle intègre plusieurs pratiques dans un cadre structuré.

Les architecture orienté services permettent de mettre en œuvre, d’implémenter et d’exécuter des fonctionnalités métier et non métier.

On dit souvent que "SOA fournit l'agilité d'entreprise", il n’y a pas de remise en cause de l’existant lors d’évolutions technologiques. Les entreprises devant s’adapter en permanence et être de plus en plus réactives, SOA permet de développer de nouvelles fonctions rapidement et de répondre aux besoins changeants de l’entreprise et des clients. C’est l’activité qui doit piloter la technologie et non l’inverse.

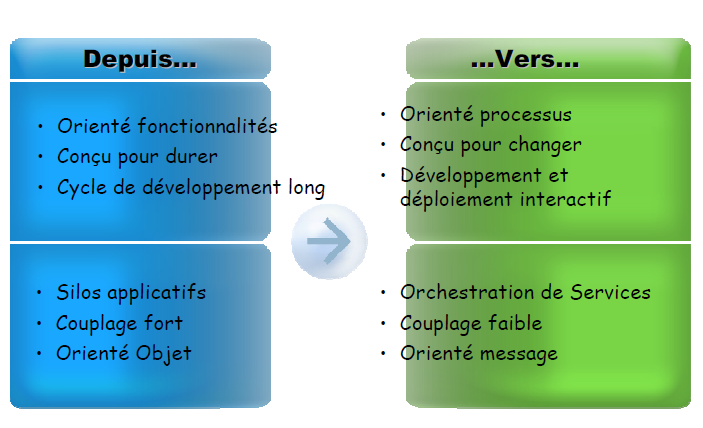
**Avantages :**

* Couplage faible entre les services, l’indépendance par rapport aux aspects technologiques et la mise à l’échelle. La propriété de couplage faible implique qu’un service n’appelle pas directement un autre service.
* Flexibilité : SOA vous donne la possibilité d'écrire divers composants de votre architecture dans la langue et la plate-forme de votre choix. Cela signifie que vous pouvez écrire le côté client dans un langage plus dynamique et productif comme Python / Ruby / Javascript et écrire les composants critiques dans les langages de bas niveau comme Java ou C.
* Test et débogage plus faciles : en isolant vos composants dans différents services, il est facile de les tester et de les déboguer individuellement. De nombreuses organisations ont des équipes distinctes pour développer, tester et entretenir ces composants.
* Evolutivité : Cela va de pair avec la flexibilité. Le fait de disposer de composants séparés simplifie considérablement l'évolution de votre architecture. Vous pouvez également mettre à l'échelle un composant particulier et le tester de manière isolée, sans affecter les autres.
* Réutilisation : Puisque divers composants sont construits séparément, il devient beaucoup plus facile de les réutiliser plus tard.

**Inconvénients SOA :**

* Difficulté d’effectuer la gestion des priorités et avec des performances faibles : les SOA ont une performance réduite pour les traitements simples, car ils sont constitués de couches métier supplémentaire.

Evolution monolithique vers SOA :



1. Microservices

**Une évolution des SOA ?**

On parle d’évolution puisque les microservices ont de nombreux points communs avec le SOA.

Ils sont tous deux dédiés à la séparation des modules grâce à des services.

Les microservices et le SOA ont cependant plusieurs différences, notamment au niveau de la définition des services, des méthodes, de leur création, mais aussi de leur étendue et de leurs principes. Il est intéressant de noter que parmi les entreprises qui utilisent actuellement les microservices, certaines, comme Netflix, considèrent qu’elles font du SOA. Les différences entre ces deux approches étant faibles, il est possible de dire que les microservices sont une manière particulière de faire du SOA.

**Les caractéristiques :**

#### Concentré sur une seule fonctionnalité :

Un microservice implémente et est responsable d’une seule fonctionnalité dans tout le système. Une fonctionnalité peut être une fonctionnalité métier contribuant à l’objectif du système, ou une fonctionnalité technique utilisée par plusieurs fonctionnalités métier.

#### Déploiements individuels des microservices :

Chaque microservice doit être déployable indépendamment des autres. Ce point est important : si l’on veut changer un microservice en particulier ou le mettre à jour, il faut être capable de le faire sans toucher aux autres microservices ni affecter leur exécution.

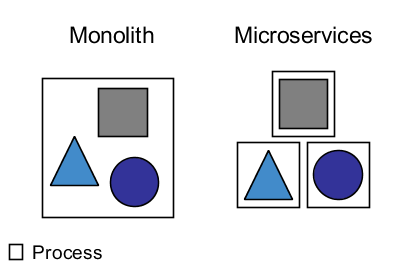
Le système doit continuer à fonctionner correctement durant la mise à jour du microservice en question, ainsi qu’après l’avoir redéployé.

#### Constitution d’un ou plusieurs processus :

Un microservice peut correspondre à un ou plusieurs processus. Il doit obligatoirement être hébergé sur des processus indépendants. En d’autres termes, il ne doit pas partager un processus avec un autre microservice.

#### La taille des microservices :

Comme l’indique le nom ”micro”, les microservices doivent être concentrés sur une seule et unique fonctionnalité, de façon que leur développement et leur maintenance puisse se faire par des équipes de petites tailles.



**Avantages :**

* Les services soient implémentés et exécutés indépendamment.
* Hétérogénéité technologique :un microservice n’a pas besoin de connaître la technologie utilisée par les autres microservices, il est donc tout à fait possible d’avoir des services n’utilisant pas les mêmes technologies. Plus de flexibilité pour subvenir aux besoins.
* Maintenance facile
* Résilience : pour chaque service contenant au moins un processus, l’interruption d’un service n’empêchera pas les autres de fonctionner.
* Testabilité : chaque microservice n’étant responsable que d’une fonctionnalité, il est plus facile de le tester ; la simplicité des services aide à leur testabilité. Des difficultés pourront cependant arriver en testant les interactions entre les différents services. Si l’API testée a été bien définie, cela ne devrait pas être un problème.
* Reduction taille des équipes de : 7 personnes maximum

**Inconvénients :**

* Lorsque le nombre de services augmente, l'intégration et la gestion de produits entiers peuvent devenir compliquées
* Les développeurs doivent mettre des efforts supplémentaires dans la mise en œuvre du mécanisme de communication entre les services
* L'architecture entraîne généralement une consommation de mémoire accrue.
* Attention aux doubles emplois

1. SOA vs Microservices

|  |  |
| --- | --- |
| SOA | Microservices |
| Un changement significatif nécessite de modifier le monolithe | Un changement significatif consiste à créer un nouveau service |
| Axé sur la réutilisation des fonctionnalités métier. Encourage le partage de composants | Essaient de minimiser le partage des composants à travers un "contexte délimité". |
| Pour la communication, il utilise Enterprise Service Bus (ESB).  Comme chaque service communique via l'ESB, si l'un des services ralentit, il risque d'encombrer l'ESB avec des demandes pour ce service. | Pour la communication utilise des systèmes de messagerie moins élaborés et simples.  Si un microservice a une erreur de mémoire, seul ce microservice sera affecté. Tous les autres microservices continueront à traiter les demandes régulièrement. |
| Prend en charge plusieurs protocoles de message.  Recommandé si voulez intégrer plusieurs systèmes utilisant des protocoles différents dans un environnement hétérogène. | Utilise des protocoles légers tels que HTTP, REST ou Thrift API  Recommandé si tous vos services sont accessibles via le même protocole d'accès à distance. |
| Type de message : synchrone : attente de connexion. | Type de message : asynchrone publish and subscribe |
| Les services SOA partagent le stockage de données.  Utilise plus souvent les bases de données relationnelles traditionnelles | Chaque microservice peut avoir un stockage de données indépendant.  Utilise des bases de données modernes et non relationnelles |
| Chaque équipe de développement doit connaître le mécanisme de communication commun. | Chaque équipe est presque totalement indépendante puisque les services peuvent fonctionner et être déployés indépendamment des autres services. |
| Gouvernance commune et normes | Gouvernance détendue, avec un accent plus marqué sur la collaboration des équipes et la liberté de choix |

Conclusion :

On ne peut pas simplement dire qu'une architecture est meilleure que l'autre. Cela dépend principalement du but de l'application que vous construisez. SOA est mieux adapté aux environnements d'applications d'entreprise complexes et de grande taille qui nécessitent une intégration avec de nombreuses autres applications. Cela étant dit, les applications plus petites ne conviennent pas à la SOA car elles n'ont pas besoin d'un composant middleware de messagerie. D'un autre côté, les microservices conviennent mieux aux systèmes Web plus petits et bien cloisonnés. De plus, si vous développez une application mobile ou Web, les microservices vous confèrent un plus grand contrôle en tant que développeur. En conclusion, nous pouvons conclure que puisqu'ils servent des buts différents - les microservices et la SOA sont en effet des types distincts d'architectures.