ÉTUDE EXPLORATOIRE ARCHITECTURE

Titre du projet			Réf du projet
Étude exploratoire d’architecture			Astra_01
Historique			
Version	Auteur	Description	Date
1.0	MESSOLO Jules		09/*02*/21

Table des matières

1. Contexte

Astra Recherche ***leader mondiale de la recherche médical***, veut relever le défi de ***la recherche*** et de ***la formation***, et souhaite se doter d'***outils de visioconférence*** et d'***outils de collaboration*** pour une coopération plus étroite avec ses partenaires extérieurs que sont les universités, hôpitaux et expert à travers le monde. L'intégrer à son infrastructure existante, qu'elle consent aussi à voire évoluer.

2. Objectifs

- Proposer une étude exploratoire des différentes architectures, avec leurs avantages et inconvénients.
- Proposer une architecture à Astra pour son infrastructure actuelle.

Le choix de l'architecture?

Bonne compréhension du métier, des contraintes fonctionnelles et non fonctionnelles (Sécurité, performance, système d'exploitation, choix des technologies,l'hébergement attendu)

Couvrir tous les différents types d'utilisateurs et leur différents mode d'accès

l'architecture logiciel va permettre de répondre aux questions suivantes : Est ce que le logiciel doit répondre rapidement? Quel volume de données traiter? Données centralisées ou reparties? Comment les utilisateurs doivent accéder à l'application du point de vue réseau? Sur quel type de matériel? Combien d'utilisateurs?

Qu'est ce qui fait une bonne architecture?

Évolutivité :

Anticiper les évolutions futures du logiciel en fonction des besoins métiers

Simplicité: sa simplicité pour éviter la dette technique,

Compréhensible: facilité sa prise en main (respecter les standards et documenter)

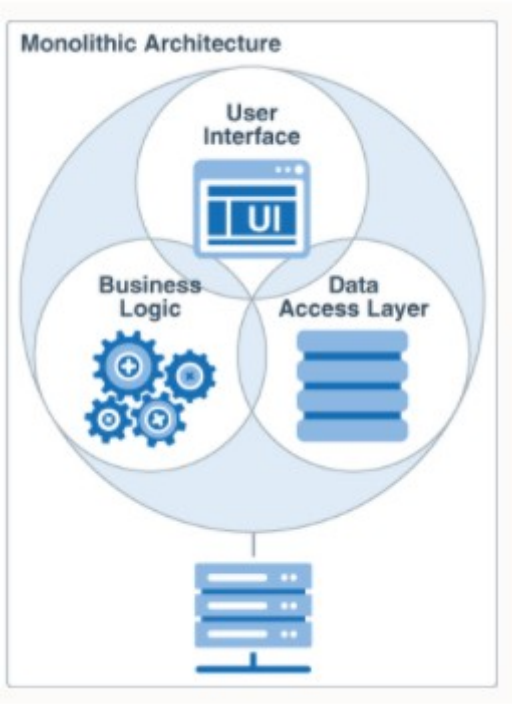
Maintenabilité: doit intégrer l'outillage nécessaire à sa maintenance

Compatibilité: doit définir la compatibilité avec les différentes plate-formes matérielles, système d'exploitation qui conviennent.

Interconnectivité: l'architecture du logiciel doit permettre son interconnectivité avec d'autre systèmes d'informations.

3. Étude exploratoire: Architecture Monolithique

Dans *une application monolithique* l'intégralité de l'application est créée en tant qu'unité unique contenant toutes les logiques métier. L*es fonctionnalités* sont construites comme une base de code unique. L'application sera déployée en tant que processus unique



Avantages

Les applications monolithiques sont généralement ***plus facile à implémenter***, à ***tester et déployer, du*** fait du faible nombre d'élément à prendre en compte***.*** Elles offrent souvent de ***meilleurs rendements*** que d'autres architectures.

C'est une architecture ***mature*** qui est utilisée depuis de nombreuse années, qui au-delà des avantages présente quelques inconvénients.

Inconvénients

Gestion des évolutions

L'architecture monolithique rend difficile l'ajout de nouvelle fonctionnalité.Chaque évolution est propice à affecter le bon fonctionnement de l'application. Donc des tests de bout en bout doivent être réalisés à chaque nouvelle implémentation. Quand au choix du langage de programmation, il s'avère définitif et ne peut s'adapter aux tendance du marché.

Gestion des erreurs et maintenance

Si une fonctionnalité est en erreur, toute l'application est à tester. Cette erreur peut entraîner l'arrêt de l'application entière.Ciblé une erreur et y remédier nécessite du temps, et expose toute l'application à de potentiel régression. On se retrouve avec une application dont la complexité augmentent avec le nombre de fonctionnalité implémenté.

Gestion de la scalabilité

Du fait de l'agglomération des ressources dans une architecture monolithique, la mise à l'échelle horizontale d'une fonctionnalité passe par la mise à l'échelle complète de l'application. Il n'est pas possible de cibler le redimensionnement. Seule la mise à l'échelle vertical est possible.

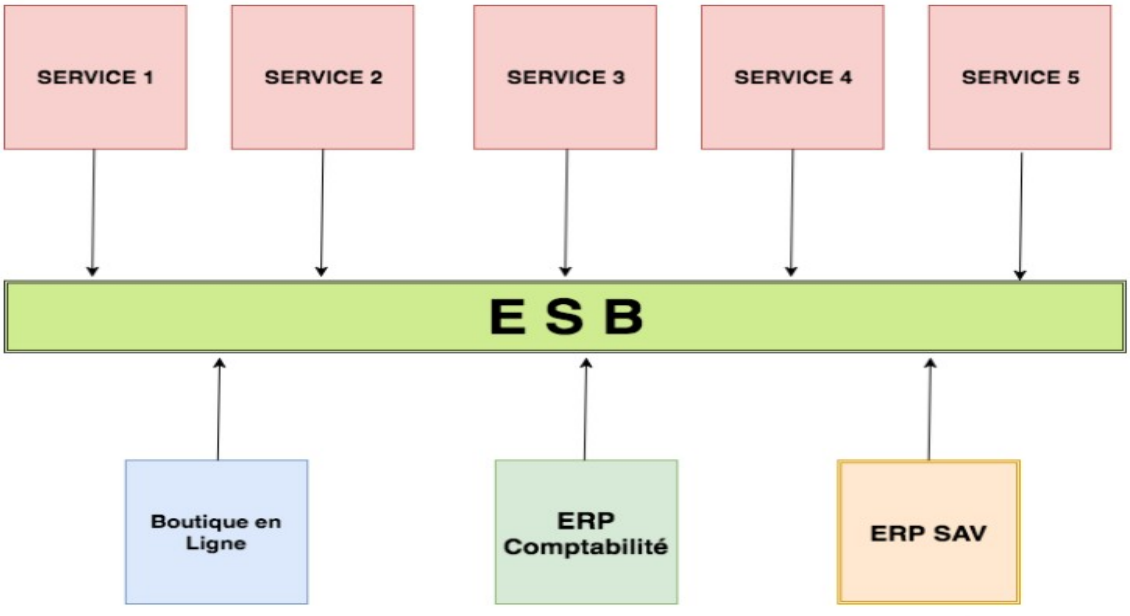
Adaptabilité

Difficile d'adapter une application monolithique déjà en place au tendance du marché, de l'adapter aux nouvelles technologies utilisés ou technique de conception.

4. Étude exploratoire: Architecture S.O.A

L'architecture orientée service (ou S.O.A, Service-Orienté Architecture) est un modèle de conception qui rend des composants logiciels réutilisables, grâce à des interfaces de services (Web services) qui utilisent un langage commun pour communiquer (XML, JSON). Ses services sont gérés séparément et contiennent des fonctionnalités (récupérer des informations ou effectuer des opérations) cohérentes et indépendantes des autres services.

Dans le modèle Web Service de l'architecture S.O.A, le langage W.S.D.L sert à connecter les interfaces aux services, et le protocole SOAP à définir les API des composants. Composant qui à l'aide d'un E.S.B (Entreprise services bus) communiquent. Ce qui à permet d'améliorer et de renforcer la gouvernance des données.



Avantages

Gestion des évolutions

S.O.A apporte une plus ***grande flexibilité***. La possibilité d'utiliser des services déjà développés ***accélère la création et l'évolution d'applications***. Les développeurs n'ont pas à recommencer de A à Z. Ce qui par cascade permet de ***réduire les coût de développement***.

Gestion des erreurs et maintenance

Les services étant indépendants, il est ***plus facile de cibler la panne***, la résoudre sans affecter le reste de l'application.Déboguer un service est plus fiable que de déboguer toute une application.

Gestion de la scalabilité

L'architecture S.O.A permet à un service de s'adapter à une montée en charge, l' E.S.B, interlocuteur commun à tous les composants pourra rediriger les requêtes vers une autre instance de ce service et permettre au service de continuer de fonctionner.

Inconvénients

Coût de développement initiaux

Plus il y a de composants, plus il faut d'interfaces et plus la conception logicielle se complique Les limites des web services du protocole SOAP qui reste difficile à mettre en place.

La lourdeur de SOAP qui utilise le langage XML, pénalise l'expansion de S.O.A. Le protocole HTTP qui domine internet, à vu l'émergence de REST et des API REST, plus léger et moins fastidieux à mettre en place.

5. Étude exploratoire: Architecture Microservice

Une architecture microservice se différencie d'une architecture monolithique de ce faite qu'elle ***décompose l'application en fonctionnalité clé***. Chacune de ses fonctionnalités est appelé service ou microservice.

Une architecture microservices en plus de reprendre les principes d'une architecture S.O.A, permet de restructurer la façon de développer une application et représente aussi une ***méthodologie de développement.***

Avantages

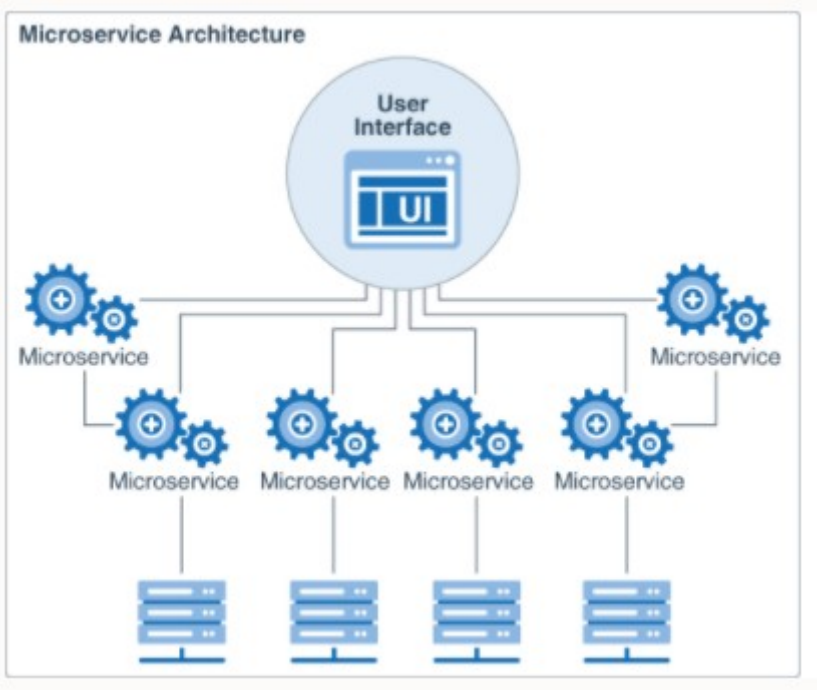
Gestion des évolutions

Chaque service peut être ***développé et déployé indépendamment*** des autres et subir des évolution qui lui sont propres. Chaque microservice peut adopter une technologie particulière.

L'ajout de nouvelles fonctionnalités devient alors plus simple et n'impacte pas la solution entière.

L'arrêt d'un service n'entrave pas le bon fonctionnement de l'application.

.



Gestion des erreurs et maintenance

L'architecture microservice permet de développer des application plus résilientes.Un service en échec n'impacte pas les autres.Permet aux équipes de se réorganiser, pour mieux s'adapter en cas de panne ou d'évolution.

Gestion de la Scalabilité

Il est plus facile d'étendre les déploiements sur plusieurs serveurs pour répondre à la monté en charge. Chaque service utilisant ses propres ressources et pouvant utiliser sa propre technologie, une mise à l'échelle horizontale de celui ci est possible.

Nouvelle méthodologie de travail

Adopter une architecture microservice permet d'adopter de nouvelles méthodes de travail. L'hétérogénéité technologique facilite le travaille des développeurs,en leur permettant de choisir leur propre outils et langage de programmation. D'être plus réactive au changement et aux tendance du marché.

Chaque équipe aura en charge le déploiement d'un service précis, et pourra adopter sa cadence.

Inconvénients

Déploiement

Architecture microservice implique un travail bien ardu lors du déploiement de chaque microservice. La gestion des pipelines de livraison, la sélection des outils nécessite une certaine expertise.

Maintenance

S'assurer que chaque microservice fonctionne correctement nécessite un travail quotidien et une familiarité avec les outils usités: Docker et Kubernetes. D'où l'importance d'adopter une culture Devops.

Défis d'une architecture microservice

Les défis que propose cette architecture sont :

- Une analyse fonctionnelle poussé pour **bien établir le périmètre de chaque microservice**. L'approche «*Domain Driven Design*» permet de faire ressortir un Domain métier, des entités qui vont participer à la définition du microservice.
- **Préserver indépendance des microservices** quelque soit la forme d'intégration de ceux ci. Ne pas introduire de couplage technologique (entre le choix de langage utilisé, le protocole de communication et le format d'échange des informations).
- La gestion des **échanges en microservices**, nécessite un point d'attention. Plus l'infrastructure intègre de microservice plus les échanges entre microservices peuvent produire de la latence. Le choix de communication synchrone et asynchrone va se poser, quel protocole d'échange privilégier ?
- Comment **tester un ensemble de microservices** ? Si l'autonomie et l'indépendance d'un microservices permet à celui d'être testé de bout en bout, il apparaît plus complexe d'écrire un test pour un ensemble de microservices. Reproduire l'environnement de

production complet pour les tests est plus compliqué. Les communications asynchrones rendent difficiles la capacité de tester un ensemble de microservices.

- Compte tenu de la difficulté de tester l'architecture microservices de bout en bout, il peut être intéressant de mettre en place une surveillance appelé 'Monitoring'. Pour déceler rapidement des anomalies .
- La gestion des déploiements. Maîtriser l'automatisation des processus menant jusqu'à la mise en production.Choisir les outils adéquats pour assurer le déploiement.
- La sécurité est un énorme enjeux pour une architecture microservice. Elle nécessite plus d'attention qu'une application monolithique du faite de la multitude de composant à sécuriser. Un ensemble de bonne pratique et une bonne stratégie de sécurité en amont peuvent permettre d'éviter les Cyberattaques (l'homme du milieu, les attaques par injection, les scripts inter-sites, D Dos et autres.)

Conclusion

Chacune des architectures présentent des avantages et des inconvénients.

L'émergence du Cloud Computing et et la modernisation rapide des applications sur internet rendent l'utilisation de ***l'architecture monolithique en perte de vitesse*** avec les tendances actuelles, bien qu'elle soit la plus simple et rapide à déployer. A coté de sa, ***la flexibilité*** sur bien des points qu'offre ***l'architecture microservice***, qui en fait l'architecture tendance et la plus adaptée du moment, s'accompagne de bien des défis quand à la *sécurité* et au bon fonctionnement de celle-ci.

Une bonne compréhension du métier, des contraintes fonctionnelles et non fonctionnelles, une vison clair des évolutions avenir permet de mieux se positionner sur l'architecture à adopter.