Projet de mise à niveau des outils de collaboration

Etude exploratoire d’architecture

Auteur(s) et contributeur(s)

Nom & Coordonnées	Qualité & Rôle	Société
Gérald ATTARD	Architecte logiciel	Astra

Historique des modifications et des révisions

N° version	Date	Description et circonstance de la modification	Auteur
1.0	24/05/2022	Création du document	Gérald ATTARD

Validation

N° version	Nom & Qualité	Date & Signature	Commentaires & Réserves
1.0	Terry Strasberg CTO		

Tableau des abréviations

Abr.	Sémantique
API	Application Programming Interface (trad. *interface de programmation d'application*)
CRM	Customer Relationship Management (trad. *gestion de la relation client)*
ERP	Enterprise Resource Planning (trad. *progiciel de gestion intégré*)
HTTP	HypertexT Transfer Protocol (trad. *protocole de transfert hypertexte*)
IRA	Institut de Recherche Astra
PI	Propriété Intellectuelle
REST	Representation State Transfer (trad. *transfert d'état représentatif*)
SOA	Service Oriented Archtecture (trad. *architecture orientée services*)
SSL	Secure Sockets Layer (trad. *couche de sockets sécurisée*)
TLS	Transport Layer Security (trad. *sécurité de la couche de transport*)

Table des matières

Contexte professionnel

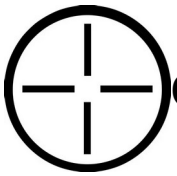
Leader dans le domaine de la recherche médicale depuis plusieurs années, l'IRA (Institut de Recherche Astra) s'est forgée une solide réputation basée sur la qualité de ses recherches médicales et la pertinence des résultats mis en évidence.

Ses activités ont toujours été accompagnées d'une forte propension à la coopération avec différents organismes, organisations et partenaires externes répartis à travers le monde.

Associée à ce contexte coopératif international, la croissance de l'IRA ,en terme d'effectif, s'est accompagnée par une augmentation des frais de déplacement, impactant financièrement certains projets.

Ainsi, au travers du développement rapide des technologies de l'information, et en tenant compte des réglementation internationales en vigueur autant politiques qu'industrielles, l'IRA se doit d'adapter ses outils collaboratifs afin de répondre à de nouveaux défis.

Ces challenges devront alors prendre en considération aussi bien l'infrastructure existante que les besoins actuels et prévisionnels, pour imaginer les outils collaboratifs de demain en accord avec les standards éthiques et moraux de l'Institut.



Objectif du document

Ce document a pour ambition de décrire les résultats des recherches sur les options d'architecture, les avantages et inconvénients, et les options d'optimisation.

Il présentera une analyse des architectures possibles pour la mission relative au contexte décrit ci-dessus.

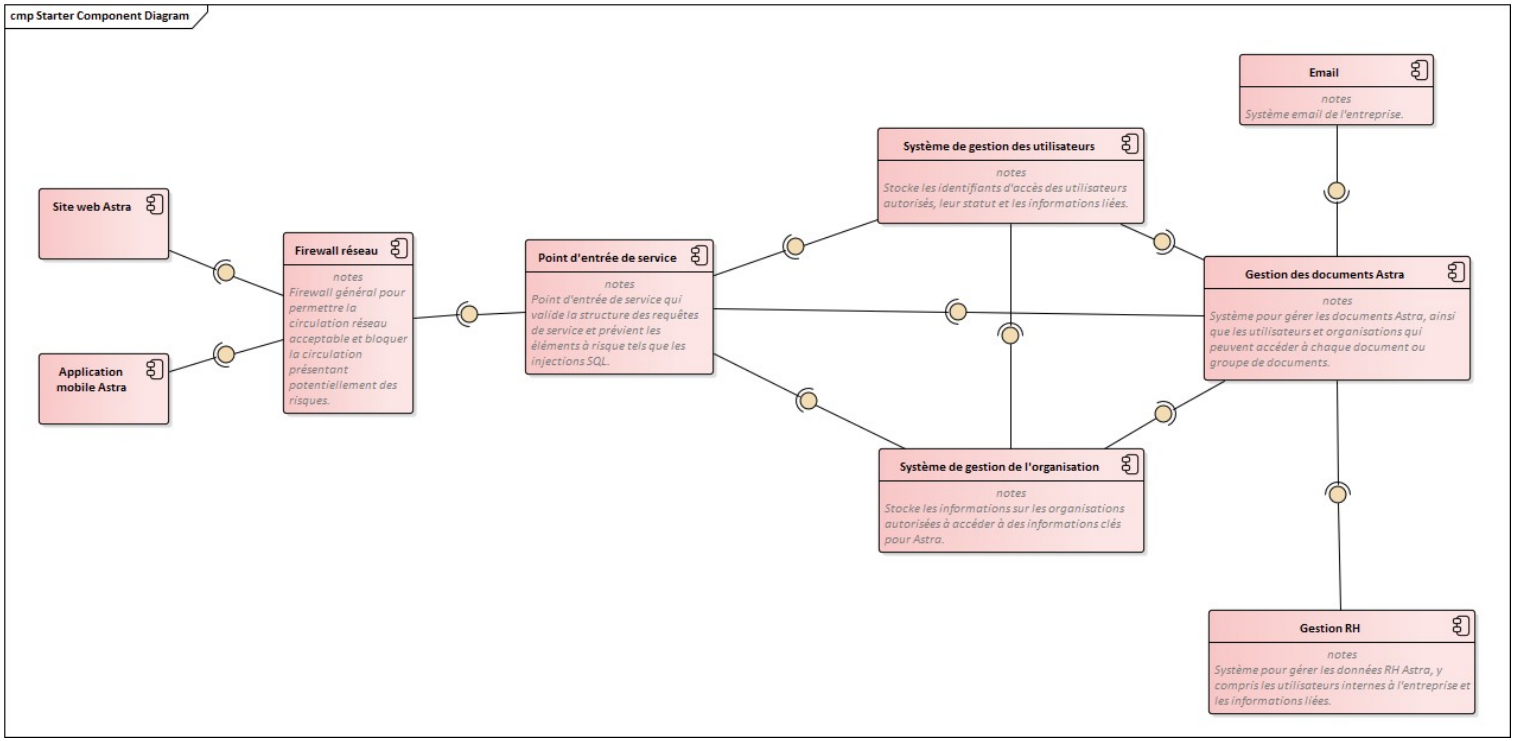
L'analyse sera orientée sur les aspects techniques et le document motivera une conclusion en justifiant la meilleure option.

Ainsi, ce document présentera :

- différentes options d'architecture ;
- les avantages et inconvénients de chaque option d'architecture détaillés ;
- les options de sécurité relatives à la protection des données ;
- les options d'intégration des applications au format web et mobiles.

Existant - Baseline Business Architecture

En tenant compte du contexte décrit précédemment, il est important, dans un premier temps, de prendre en compte l'architecture existante de l'IRA. Ainsi, le diagramme de composants ci-dessous représente cette architecture :



Relativement au diagramme présenté ci-dessus, et de par le manque d'élément présenté, nous conviendrons arbitrairement que l'architecture d'Astra se base sur un modèle d'architecture de type Client-Serveur.

Nous pouvons constater dans le diagramme ci-dessus, la présence de plusieurs composants, tels que :

- le **site web Astra** affiche des informations publiques sur l'entreprise de même que des informations protégées par login basées sur l'organisation et le rôle de l'utilisateur. Il est construit comme une application web réactive permettant l'accès depuis une variété d'appareils et de tailles d'écran ;

- l'**application mobile Astra** est une application mobile pour les appareils Android et iOS permettant aux utilisateurs mobile d'accéder aux informations Astra autorisées par leur login et leur rôle d'utilisateur, depuis un appareil mobile. Elle dispose d'un stockage limité de documents et d'autres fonctionnalités spécifiques à une application mobile au-delà de ce qui est permis par le site web ;

- le **firewall réseau** configuré pour protéger les systèmes Astra de la circulation réseau inattendue ou non planifiée. Il fournit l'accès via le port 80 aux systèmes et services exposés alors que les systèmes internes peuvent utiliser différents ports HTTP pour la protection des données ;

- le **point d'entrée de service** est un dispositif qui vérifie que les utilisateurs accèdent uniquement aux services auxquels ils ont accès ;

- le **système de gestion des utilisateurs** gére les utilisateurs ayant la permission d'accéder aux services et à d'autres systèmes internes, dont le rôle, l'authentification, et les capacités liées aux utilisateurs ;

- le **système de gestion de l'organisation** gère les organisations ayant accès aux données et services Astra. Les utilisateurs doivent alors appartenir à une organisation autorisée. Certains services permettent à tout utilisateur d'une organisation d'accéder à des données et documents limités ;

- le service d'**email** est service typique pour recevoir et envoyer des emails, pour les utilisateurs internes à Astra. Il gère les emails transactionnels envoyés par une API ;

- la **gestion des documents Astra** dispose de protections permettant uniquement aux utilisateurs internes et externes autorisés d'accéder à des documents spécifiques, sur la base du rôle utilisateur ou en tant qu'utilisateur ayant la permission d'accéder à des documents et dossiers spécifiques ;

- la **gestion RH** est un système gérant les utilisateurs, salariés et prestataires internes à Astra, et incluant le rôle, département et les permissions d'accès de l'utilisateur.

Toutes les fonctionnalités ci-dessus s'accompagnent de standards d'architecture informatique propre à l'IRA, dont voici les principaux :

- toutes les données doivent être encryptées lors de leur transfert ;

- les utilisateurs finaux ne doivent pas pouvoir stocker des données localement sur leur ordinateur ou appareil, à l'exception des fichiers qui peuvent être mis en cache dans l'application mobile Astra. Toutes les données à partager avec d'autres utilisateurs doivent respecter les standards de partage de données et du système de gestion de contenu ;

- tous les composants internes doivent être protégés derrière un firewall et autres services de données. Les modifications de ces dispositifs peuvent être effectuées pour la solution dès lors qu'elles respectent des standards de sécurité. Les modifications recommandées devront être identifiées et passées en revue par les équipes de direction ;

- l'environnement IT est construit sur des serveurs Linux physiques et virtuels. Ainsi, tous les composants de la solution doivent être conformes à cette combinaison ;

- toutes les interfaces de service doivent être construites selon les standards de l'industrie pour l'authentification utilisateur et les meilleures pratiques de sécurité ;

- tous les utilisateurs accédant à une réunion collaborative doivent avoir un compte utilisateur au sein du système et être validés lorsqu'ils accèdent à une réunion web ;

- à l'exception des informations disponibles publiquement, tout accès à un service ou document doit être validé par les systèmes utilisateur et organisation ;

- le système d'email permettra aux systèmes authentifiés d'envoyer automatiquement des emails transactionnels par le biais d'une API. Les utilisateurs authentifiés peuvent envoyer des emails ;

- tous les accès aux services et aux composants doivent correspondre au rôle et à l'authentification de l'utilisateur ;

- tout accès à un composant doit se faire à travers le firewall ;

- tout accès à un service doit passer par le point d'entrée de service.

🎯 Objectif - Target Business Architecture

Exigences fonctionnelles

En tenant du contexte décrit en introduction, l'IRA se doit de relever divers challenges évoluant dans des environnements aussi bien business que technologique ou économique.

Ainsi, un besoin d'outils logiciels collaboratifs personnalisés a été identifié afin d'optimiser les échanges avec des partenaires externes tels que des universités, des hôpitaux, et des experts à travers le monde.

La solution devra également protéger :

- la Propriété Intellectuelle (PI) d'Astra ;
- les recherches d'Astra ;
- l'historique médical de toute personne participant aux recherches.

Plus spécifiquement, Astra devra se doter d'outils de visioconférence afin de collaborer en temps réel avec ses partenaires externes ; ces interactions devront être basés sur des technologies web.

Malgré plusieurs essais d'outils commerciaux existants, ceux-ci n'ont pas répondu aux contraintes sécuritaires propres à l'IRA. La nouvelle solution devra assurer un haut niveau de protection et de confidentialité des données de l'Institut.

En outre, Astra devra être capable de contrôler et maîtriser pleinement cette nouvelle solution personnalisée pour :

- garantir que seuls les participants d'une réunion autorisés assistent aux réunions en ligne ;
- protéger la PI d'Astra ;
- assurer la confidentialité des informations privées des sujets de recherches.

Exigences techniques

Afin de pouvoir piloter le planning de mise en œuvre de la solution d'architecture logicielle adoptée, il est nécessaire de lister les exigences techniques auxquelles devra se conformer la solution.

Vous trouverez ci-dessous une liste des exigences techniques initiales définies pour la solution. Celles-ci constituent les conditions requises de haut niveau business, pensées pour piloter le planning produit et architectural initial. Ces conditions requises couvrent les fonctionnalités de haut niveau suivantes :

- des réunions web interactives entre de multiples participants avec audio et vidéo en streaming entre tous les participants ;
- la diffusion de contenu audio et vidéo ad hoc des sessions enregistrées ;
- la possibilité pour les utilisateurs authentifiés de rechercher des sessions et des cours enregistrés ;
- la possibilité pour les utilisateurs authentifiés de participer aux réunions web pour lesquelles ils ont l'autorisation.

Pour répondre aux fonctionnalités exprimées supra, l'architecture envisagée devra répondre à plusieurs exigences techniques priorisées de 1 à 3 dans les paragraphes ci-dessous, selon la sémantique suivante :

- Priorité 1 : exigences requises ;
- Priorité 2 : exigences préférées mais non requises ;
- Priorité 3 : exigences informelles.

Priorité 1

- Gérer des réunions web interactives en temps réel avec streaming vidéo et audio pour jusqu'à 10 participants.
- Gérer les présentations à des classes avec un grand nombre de participants, excédant 500 personnes.
- Gérer les ordinateurs fixes et portables sous Windows, Mac OSX, et Linux.
- Gérer les navigateurs Chrome.
- Gérer les appareils iOS, aussi bien les tablettes que les téléphones.
- Permettre des améliorations et modifications futures du système.
- Fournir une méthode aux utilisateurs pour trouver les réunions, enregistrements, et documents auxquels ils ont accès.

Priorité 2

- Enregistrer les réunions web et *streamer* les enregistrements à la demande.
- Fournir une méthode aux participants aux cours pour poser des questions sans interrompre la présentation à la classe.
- Gérer les navigateurs Firefox.
- Gérer les appareils Android, aussi bien les tablettes que les téléphones.

Priorité 3

- Gérer des réunions en temps réel avec jusqu'à 100 participants.
- Gérer les navigateurs Safari.

Exigences opérationnelles

Relativement aux fonctionnalités exprimées, il faudra également que la nouvelle solution préconisée réponde à des exigences opérationnelles, afin de garantir son intégration au sein de l'environnement de production informatique d'Astra.

Ces exigences opérationnelles sont priorisées de 1 à 2 dans les paragraphes ci-dessous, selon la sémantique suivante :

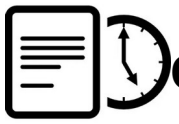
- Priorité 1 : exigences requises ;
- Priorité 2 : exigences préférées mais non requises.

Priorité 1

- Toute communication, y compris le streaming de médias et le partage de documents, doit être sécurisée pour ne permettre l'accès qu'aux utilisateurs autorisés.
- Toutes les données doivent être cryptées sur le réseau avec SSL, TLS, ou des technologies similaires.
- Tout composant interne doit être protégé derrière un firewall et d'autres dispositifs de données.
- Les utilisateurs accédant à une présentation ou session de formation web doivent répondre aux exigences d'authentification utilisateur pour la présentation ou session de formation.
- Les utilisateurs doivent être authentifiés selon les systèmes de gestion des utilisateurs ou de l'organisation.

Priorité 2

- Fournir une méthode pour permettre aux utilisateurs invités l'accès aux documents, enregistrements, ou réunions marqués comme accessibles aux invités.



Contraintes fonctionnelles

En ce qui concerne l'élaboration de ce projet, celui-ci devra répondre aux contraintes suivantes :

- le projet initial est approuvé pour un coût de 50 000 $ et une période de six mois pour définir l'architecture et préparer un projet subséquent de développement d'un prototype ;
- l'architecture peut être définie comme composants à développer ou déjà développés ;
- une justification des coûts initiaux et prévisionnels comparés à une solution sur étagère sera nécessaire, pour les composants à développer ;
- l'architecture doit correspondre à un scénario de valeur optimale par rapport aux coûts.

Lignes directrices

Les orientations listées ci-dessous seront à suivre préférentiellement afin d'obtenir un système d'information propres aux attentes d'Astra :

- les solutions open source sont préférées aux solutions payantes ;
- le support continu des composants choisis doit être pris en compte lors de la sélection des composants ou des décisions de développer ou acheter ;
- toute solution déjà développée ou open source doit s'intégrer à une pile technologique commune lorsque c'est possible pour réduire les coûts de support et de maintenance continus.

Modèles d’architecture

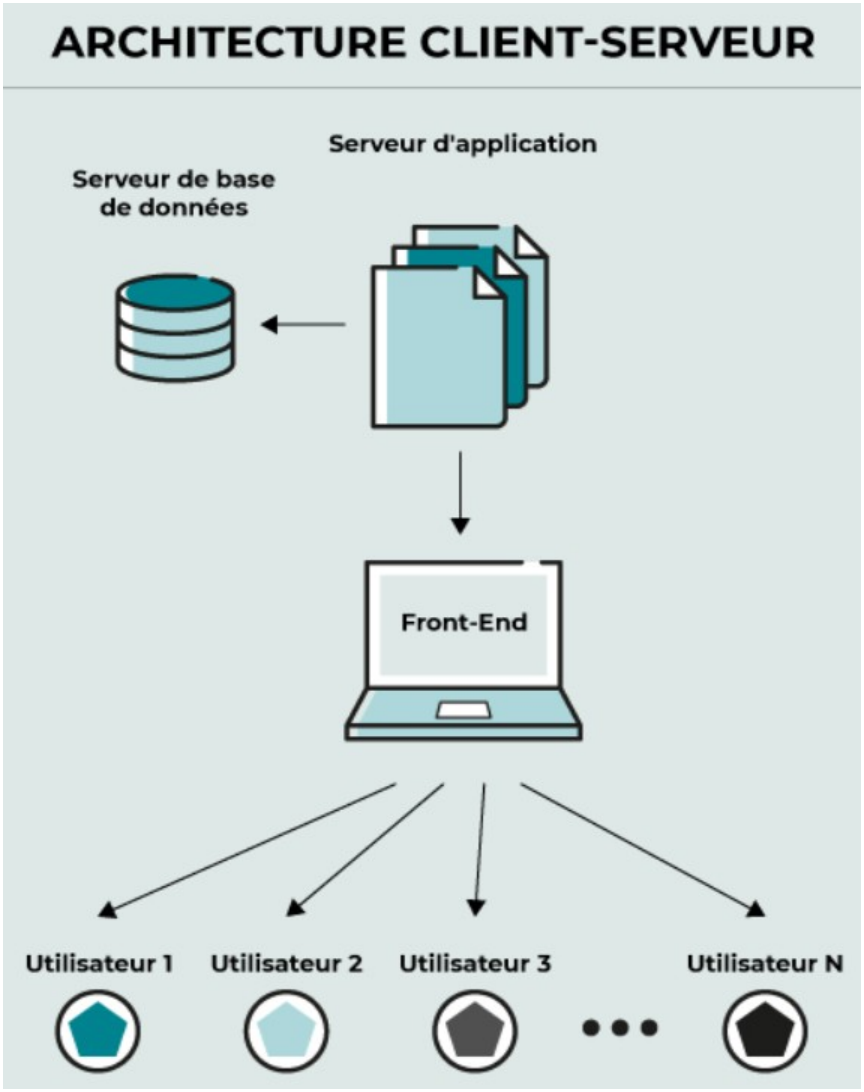
Architecture Client-Serveur

Description

L'architecture Client-Serveur fonctionne selon un principe de répartition des tâches entre les fournisseurs d'un service, appelés **Serveurs**, et les consommateurs du service, appelés **Clients**.

Ainsi, via un dispositif quelconque (ordinateur, tablette, smartphone), un **Client** peut demander un service à un **Serveur.** Le dispositif sert alors de médiateur entre le consommateur de service (le **Client**) et le fournisseur de service (le **Serveur**), et sait comment parler avec les deux.

Le diagramme ci-dessous décrit ce processus :



Dans le schéma précédent, il est possible de dénombrer trois parties principales :

- Le **Front-End :** il s'agit de la partie du logiciel qui interagit avec les utilisateurs, même s'ils se trouvent sur des plateformes différentes avec des technologies différentes. Dans une architecture Client-Serveur, les modules *front-end* sont conçus pour interagir avec tous les appareils existant sur le marché. Cela comprend les écrans de connexion, les menus, les écrans de données et les rapports qui fournissent et reçoivent des informations des utilisateurs. Par exemple, la plupart des outils et frameworks de développement permettent de créer une version du programme qui fonctionne à la fois pour les PC, les tablettes et les téléphones.
- Le **serveur d'application (Application Server)** : il s'agit du Serveur où sont installés les modules logiciels de l'application. Il se connecte à la base de données et interagit avec les utilisateurs. Le serveur d'application est comme le cuisinier du restaurant de notre exemple.
- Le **serveur de base de données (Database Server)** : il contient les tables, les index et les données gérés par l'application. Les recherches et les opérations d'insertion/de suppression/de mise à jour sont exécutées ici.

Avantages

L'architecture Client-Serveur sépare le matériel, le logiciel et la fonctionnalité du système. Par exemple, si une adaptation du logiciel est nécessaire pour un pays spécifique, autrement dit si un changement de fonctionnalité est nécessaire, celui-ci peut être adapté dans le système sans avoir à développer une nouvelle version pour les téléphones, les tablettes ou les ordinateurs portables.

En outre, puisque cette architecture sépare le matériel, le logiciel et la fonctionnalité du système, seule la partie *front-end* doit être adaptée pour communiquer avec différents appareils.

Inconvénients

Dans le cas où énormément de Clients demanderaient simultanément des données au Serveur, celui-ci pourrait se voir surchargé, et donc cela pourrait provoquer des interruptions de services.

De plus, si ces interruptions de services sont provoquées par la panne du Serveur, alors aucun utilisateur ne pourrait plus utiliser le système.

Utilisation

Ce modèle d’architecture est particulièrement adapté pour le déploiement de solutions logicielles telles que des Progiciels de gestion intégré (ERP), des serveurs d’impression ou encore des serveurs de messagerie.

Architecture orientée évènements

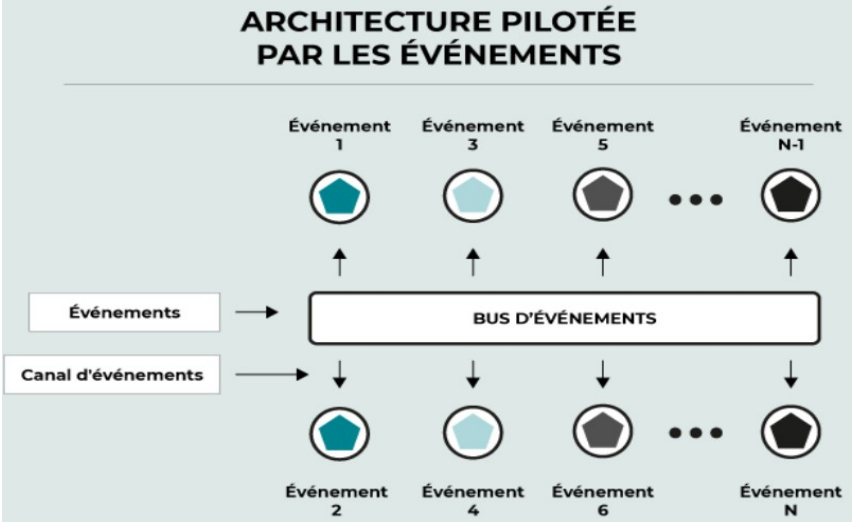
Description

Une architecture pilotée par les événements fonctionne en se basant sur le principe d'alerte, c'est à dire qu'elle produit, détecte et agit selon les événements du système pertinents pour l'utilisateur.

Si aucun événement ne se produit, rien ne se passe dans le système.

Cette architecture définit donc des événements comme déclencheurs qui, lorsqu'ils s'activent, provoquent des comportements spécifiques, comme l'envoi d'une alerte.

Ce modèle d'architecture est représenté par le schéma suivant :



Dans le schéma précédent, il est possible de dénombrer trois parties principales :

- Le **bus d'événements (Event Bus) :** il s'agit d'une ligne de communication qui relie tous les utilisateurs aux événements. Cette ligne n'est pas un câble physique mais fait référence à une connexion logique, similaire à celle qui existe entre votre ordinateur portable et un site web. Les événements sont transportés par ce bus d'événements abstrait et atteignent tous les utilisateurs du système. Le bus n'existe pas réellement, mais il sert à décrire l'échange de données entre de nombreux composants ou utilisateurs.
- Le **canal d'événements (Event Channel) :** Un canal d'événements est une étiquette pour le type d'événements auxquels les utilisateurs s'abonnent. Si un utilisateur est abonné à un canal donné, il recevra tous les messages qui s'y rapportent.
- Le **traitement des événements (Event Processing) :** Toutes les actions entreprises *après* un événement donné sont exécutées dans le module de traitement des événements. Ce module agit selon un événement donné pour servir l'utilisateur.

Avantages

Lorsque des utilisateurs ont besoin d'écouter (de s'abonner à) différents messages sur un sujet spécifique, l'architecture pilotée par les événements est la bonne approche.

Cela se produit lorsque le système doit réagir à des événements sans comportement prédictif : par exemple, si je m'abonne à la chaîne « *Informations africaines* », je n'ai aucun contrôle sur les actualités que je recevrai ni sur leur quantité.

Ici, le système n'est pas prédictif ; il réagit aux événements générés par d'autres utilisateurs.

En outre, lorsque ces utilisateurs sont en dehors de l'organisation, les événements peuvent être transportés dans un bus d'événements publics et atteindre tous les utilisateurs immédiatement.

Le nombre d’événements pouvant être traités en même temps par ce modèle d’architecture n’est pas limité quantitativement.

Inconvénients

Lorsqu’un nombre important d’évènement peuvent être survenir, et qu’ils surviennent simultanément, alors le bus d'événements peut être surchargé et occasionner des interruptions du service d’abonnement.

Autrement dit : si le bus d'événements tombe en panne, tout le système tombe en panne et comme il n'y a aucun dispositif de contrôle du flux d'événements, de nombreux événements peuvent se produire en même temps, créant alors un véritable chaos pour les utilisateurs.

Utilisation

Ce modèle d’architecture est particulièrement adapté pour le déploiement de solutions logicielles telles que des systèmes de micro-blogging, d’approvisionnement ou d’automatisation d’usine.

Architecture centrée sur les données

Description

Une architecture de données est une structure des données et des ressources d'une entreprise. Elle regroupe les modèles, les règles, les politiques et les standards autour de la collecte, du stockage, de l'intégration et de l'utilisation des données dans l'entreprise. Il s'agit donc d'un processus de standardisation.

Le but d'une architecture centrée sur les données est de permettre à chaque équipe de l'entreprise d'accéder aux données dont elle a besoin, quand elle en a besoin, et de les aider à donner un sens à ces données.

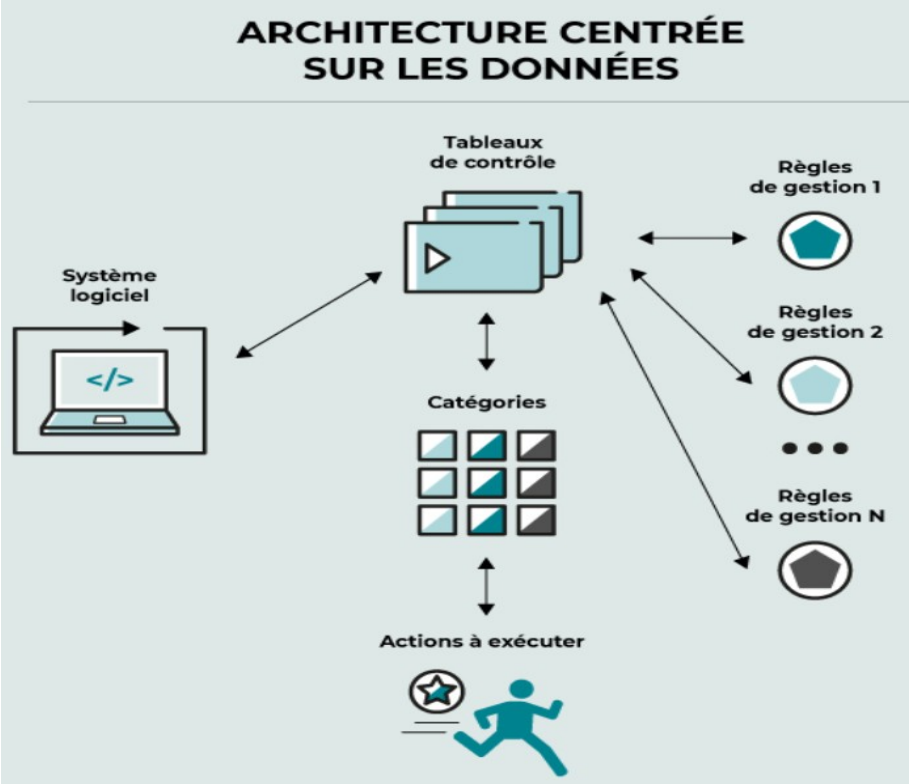
Il s'agira donc de permettre aux décideurs stratégiques d'accéder aux données librement sans avoir à demander l'aide des techniciens.

Paradoxalement, l'objectif est aussi de favoriser la collaboration entre ces deux expertises distinctes.

Cette collaboration permet donc de déterminer quelles sont les données nécessaires pour stimuler la croissance, comment collecter ces données, et comment les distribuer.

Avec l'essor du Cloud, permettant de gagner en élasticité et de réduire les coûts, l'architecture moderne centrée sur les données a pu se développer afin d'optimiser les rendements des entreprises.

Le schéma ci-dessous illustre les processus généraux de cette architecture :



Dans le schéma précédent, il est possible de dénombrer cinq parties principales :

- Le **système logiciel (Software System) :** Le système développé en utilisant le modèle d'architecture centré sur les données ;
- Les **tableaux de contrôle (Control Tables) :** Un ensemble de tableaux qui définissent les actions que le système doit exécuter pour chaque catégorie ;
- Les **catégories (Categories) :** Un ensemble de catégories d'un élément dans le système (types de clients, types de produits, types d'articles, types d'employés) utilisé pour exécuter des actions ;
- Les **actions à exécuter (Action Items) :** Un ensemble d'actions à exécuter pour une certaine catégorie ;
- Les **règles de gestion (Business Rules) :** Les règles de gestion qui déterminent les actions à exécuter en fonction des catégories.

Avantages

En cas d'évolution du système, lorsque des catégories changent ou sont ajoutées, il n'est alors pas nécessaire de modifier le code.

De plus, les règles de gestion sont beaucoup plus faciles à gérer dans un tableau que dans un ensemble de programmes, cela apporte de la clarté lorsqu'un système d'information présente une certaine complexité.

Inconvénients

La représentation graphique de cette architecture laisse tout de même transparaître des problèmes de performance. En effet, à chaque exécution du code, le système doit renouveler ses requêtes de recherche des données au sein de la base de données ; ce processus peut engendrer des ralentissements du système global.

En outre, dans le cas où la table de contrôle se verrait endommagée, l'ensemble du système ne fonctionnerait tout simplement plus.

Utilisation

Ce type d'architecture permet aux organisations de se préparer stratégiquement pour évoluer rapidement et tirer profit des opportunités liées aux technologies émergentes.

Son but est aussi de traduire les besoins de l'entreprise en besoins de données et systèmes informatiques. Elle simplifie donc l'alignement du département informatique avec l'activité.

L'architecture de données permet de gérer la diffusion d'informations et de données complexes à travers l'entreprise. L'organisation peut donc gagner en agilité.

Ce modèle d'architecture se rencontre au sein des systèmes CRM (gestion de la relation client) et des modules de paiement de commissions.

Architecture en couches

Description

La conception de logiciels nécessite de recourir à des bibliothèques.

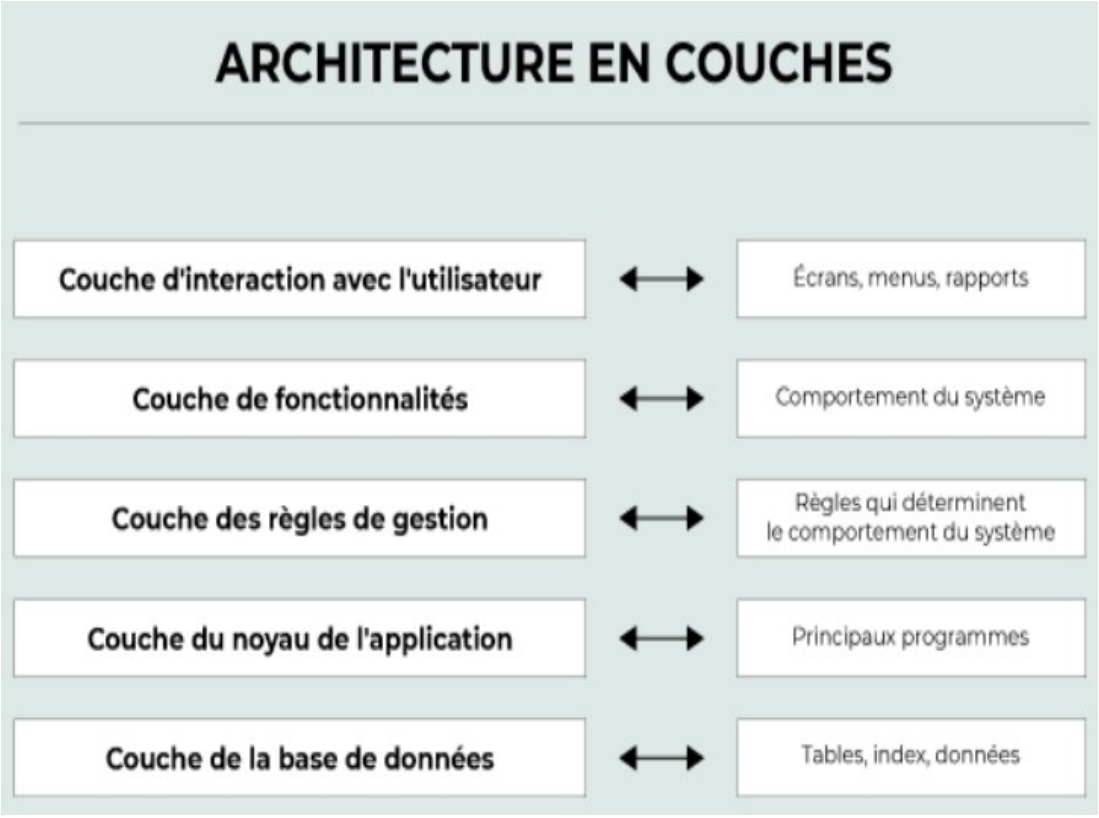
Une bibliothèque très spécialisée utilise des bibliothèques moins spécialisées qui elles-mêmes utilisent des bibliothèques génériques.

De plus, le développement efficace de composants réutilisables nécessite de créer une bibliothèque logicielle ; l'architecture en couches est la conséquence inéluctable d'une telle approche.

En effet, les nouveaux composants utilisent les anciens et ainsi de suite, la bibliothèque tend donc à devenir une sorte d'empilement de composants.

La division en couches consiste alors à regrouper les composants possédant une grande cohésion (sémantiques semblables) de manière à créer un empilement de paquetages de composants ; tous les composants des couches supérieures dépendants fonctionnellement des composants des couches inférieures.

Le schéma ci-dessous montre une version simplifiée du modèle d’architecture en couches :



Dans le schéma précédent, il est possible de dénombrer cinq parties principales :

- La **couche d'interaction avec l'utilisateur (User interaction layer) :** c'est la couche qui interagit avec les utilisateurs par le biais d'écrans, de formulaires, de menus, de rapports, etc. C'est la couche la plus visible de l'application. Elle définit l'aspect de l'application ;
- La **couche de fonctionnalités (Functionality Layer) :** il s'agit de la couche qui présente les fonctions, les méthodes et les procédures du système selon la couche des règles de gestion. Elle détermine comment les menus déroulants fonctionnent, comment les boutons fonctionnent et comment le système navigue entre les écrans ;
- La **couche des règles de gestion (Business rules layer) :** cette couche contient des règles qui déterminent le comportement de l'ensemble de l'application, par exemple : « Si une facture est imprimée, il faut envoyer un courriel au client, sélectionner tous les articles vendus et diminuer leur stock dans le module de gestion des stocks. » ;
- La **couche du noyau de l'application (Application core layer) :** cette couche contient les principaux programmes, les définitions du code et les fonctions de base de l'application. Les programmeurs travaillent la plupart du temps sur cette couche ;
- La **couche de la base de données (Database layer) :** cette couche contient les tables, les index et les données gérées par l'application. Les recherches et les opérations d'insertion/suppression/mise à jour sont exécutées ici.

Avantages

Chaque couche est autonome et indépendante des autres, c’est à dire que les changements effectués sur une couche n'affectent pas les autres. Cela s’avère relativement puissant dans le cas d’une augmentation des fonctionnalités d'une couche sans avoir à réécrire toute l'application.

De plus, de par leurs spécialisations, les couches permettent une meilleure personnalisation du système.

Inconvénients

Toujours dans le cas d’une modification du système d’information concerné, les couches peuvent rendre celui-ci plus difficile à maintenir. Ainsi, chaque changement nécessite une analyse des impacts occasionnés par ces changements.

De plus, les couches peuvent affecter les performances des applications car elles créent une surcharge dans l'exécution et donc de la lourdeur du système global : chaque couche des niveaux supérieurs doit se connecter à celles des niveaux inférieurs à chaque opération du système.

Utilisation

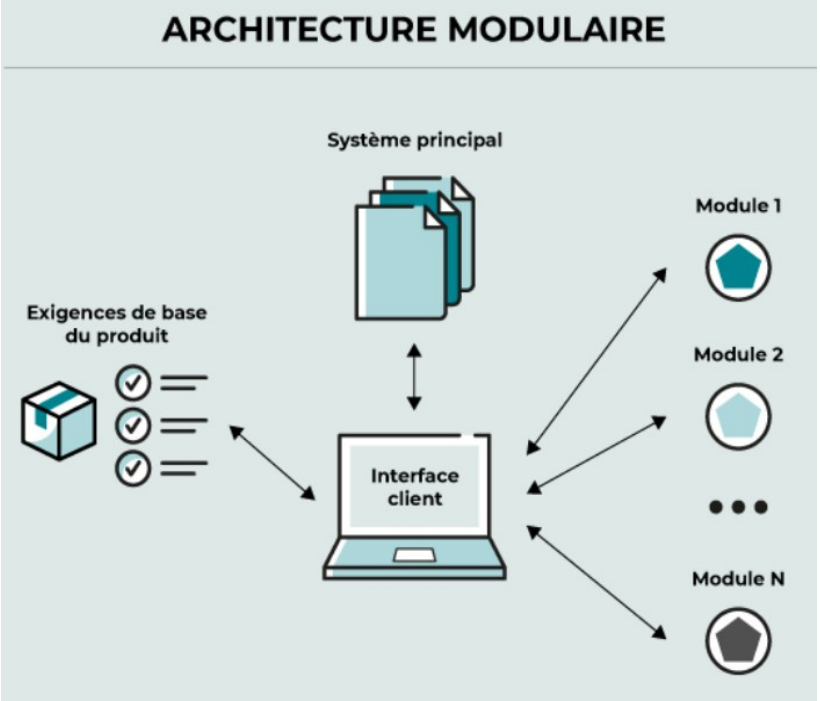
Ce modèle d’architecture se rencontre, par exemple, au sein des systèmes système de contrôle des véhicules autonomes et dans les sites web des banques en ligne.

Architecture modulaire

Description

Une telle architecture se base sur un noyau applicatif pouvant être enrichi fonctionnellement par des modules applicatifs et des extensions supplémentaires.

Ainsi, en partant d'exigences fonctionnelles « de base », il est alors possible d'étendre le système d'information global en ajoutant de nouvelles fonctionnalités sous la forme de module additifs (*plugin*) indépendants les uns des autres, comme le montre le schéma ci-dessous :



Dans le schéma précédent, il est possible de dénombrer quatre parties principales :

- Les **exigences de base du produit (Baseline product requirements) :** il s'agit de l'ensemble des exigences minimales qui définissent l'application, déterminées au début du processus de développement lorsqu'un ensemble initial de fonctionnalités a été inclus dans le produit ;
- Le **système principal (Main System) :** il s'agit de l'application à laquelle on connecte les modules. Le système principal doit fournir un moyen d'intégrer les modules et, par conséquent, il modifiera légèrement le produit de base original pour assurer la compatibilité ;
- L'**interface client (Customer Interface) :** il s'agit de la partie qui interagit avec le client, par exemple, un navigateur web (Chrome, Mozilla, etc.) ;
- Les **modules (Plug-in) :** il s'agit de modules complémentaires qui complètent les exigences minimales de l'application et lui confèrent des fonctionnalités supplémentaires.

Avantages

L'architecture modulaire est le meilleur moyen d'ajouter une fonctionnalité à un système qui n'a pas été conçu initialement pour cela, sans interagir avec le reste de l’application.

Ainsi, cette architecture supprime les limites de quantité de fonctionnalités qu'une application peut offrir. Il est alors possible d’ajouter une infinité de modules (le navigateur Chrome dispose de centaines de modules appelés *extensions*).

Inconvénients

De par la diversité des modules pouvant être ajoutés, ceux-ci peuvent devenir source de programme malveillant, laissant la porte ouverte aux virus et autres attaques venant d'acteurs externes.

En outre, cette multiplicité de modules dans une application peut également affecter ses performances, y ajoutant lourdeur et complexité...

Utilisation

Ce modèle d’architecture se rencontre, par exemple, au sein des logiciels de chiffrement ou des modules d’informations sur les sites web locaux.

Architecture orientée service

Description

L’architecture orientée services (SOA) est un modèle de conception largement utilisé, regroupant des services modulaires qui « *dialoguent* » entre eux pour prendre en charge les applications et leur déploiement par l’intermédiaire d’un service Bus.

Les files d’attente de messages assurent donc la coordination entre ces applications distribuées. Elles peuvent considérablement simplifier le codage des applications découplées, fluidifier les pics de charge, tout en améliorant les performances, la fiabilité et l’évolutivité.

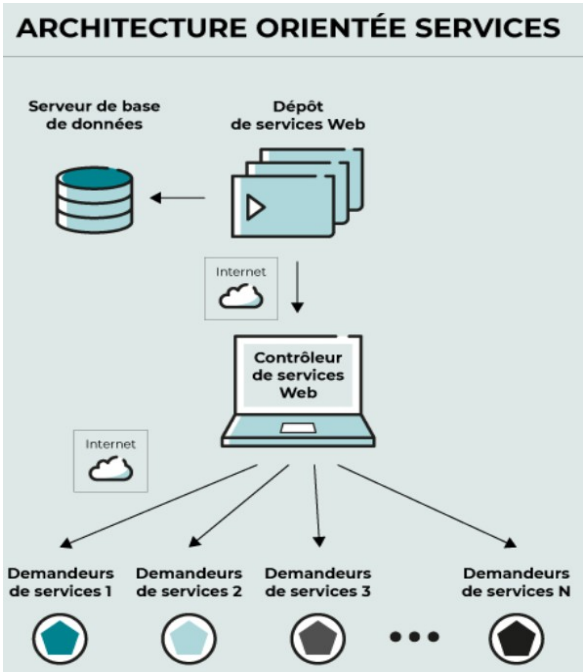
Cette architecture est une forme d'architecture de médiation qui est un modèle d'interaction applicative mettant en œuvre des **services** (composants logiciels) :

- avec une forte cohérence interne (par l'utilisation d'un format d'échange pivot, le plus souvent XML ou JSON) ;
- des couplages externes « lâches » (par l'utilisation d'une couche d'interface interopérable, le plus souvent un service web).

Ce terme, apparu dans les années 2000 concernait à l'origine essentiellement les problématiques d'interopérabilité syntaxique en relation avec les technologies d'informatique utilisées en entreprise.

Cette conception a évolué pour désigner maintenant le sous-ensemble particulier d'architecture de médiation en fonction de la technologie disponible.

Le schéma ci-dessous représente une architecture orientée service :



Dans le schéma précédent, il est possible de dénombrer quatre parties principales :

- Le **dépôt de services web (Web service repository) :** il s'agit d'une bibliothèque de services web conçue pour répondre à des demandes d'informations externes. L'information fournie est généralement un petit élément, comme un numéro, un mot, quelques variables, etc. Par exemple, un numéro de vol, un numéro de suivi de colis, le statut d'une commande (une lettre), etc. Cette bibliothèque est généralement documentée de manière très détaillée, car des applications externes font appel aux fonctions qu'elle contient ;
- Le **contrôleur de services web (Web service controller) :** ce module communique les informations contenues dans le dépôt de services web aux demandeurs de services. Lorsqu'un demandeur de service externe appelle une certaine fonction du dépôt de services web, le contrôleur de services web interprète la demande et recherche la fonction dans le dépôt de services web. Il exécute ensuite cette fonction et renvoie une valeur au demandeur ;
- Le **serveur de base de données (Database Server) :** ce serveur contient les tables, les index et les données gérés par l'application. Les recherches et les opérations d'insertion/suppression/mise à jour sont exécutées ici ;
- Les **demandeurs de services (Service Requester) :** il s'agit d'applications externes qui demandent des services au dépôt de services web par l'intermédiaire d'Internet, comme une organisation demandant des informations sur les vols à une compagnie aérienne, ou une autre entreprise demandant à un transporteur la localisation d'un colis à un moment donné.

Avantages

Ce modèle permet de collaborer avec des acteurs externes sans les laisser accéder à nos systèmes. Ainsi, les fournisseurs de services permettent aux clients d’obtenir les informations nécessaires aux services qu’ils proposent, tout en se réservant l’accès à son système interne.

Ainsi, les entreprises peuvent partager avec le monde entier, de manière ordonnée et contrôlée, la sélection des fonctions qu’elles choisissent de laisser à disposition de leurs clients.

Inconvénients

Les services web peuvent représenter une faiblesse au niveau du site pour les pirates qui veulent engorger le système. Certaines formes d'attaques sont des « dénis de service ». Elles consistent à demander le même service web des millions de fois par seconde, jusqu'à ce que le serveur tombe en panne. Il existe aujourd'hui une technologie permettant de résoudre ce problème, mais c'est toujours un problème à prendre en compte dans les architectures de services web.

Le propriétaire du service web aide d'autres sites, mais reçoit une petite rémunération pour ce faire.

Utilisation

Ce modèle d’architecture se rencontre, par exemple, au sein des logiciels de suivi de colis (PTMS), d’informations sur les vols ou encore au sein des convertisseurs de devises.

Variante : Architecture de Microservices

Description

Une architecture Microservices a pour objet de diviser une application en fonctionnalités encapsulées au sein de services autonomes. Chacun de ces services est géré et évolue indépendamment des autres services.

Les Microservices peuvent être mis à jour, étendus et déployés indépendamment les uns des autres et, par conséquent, beaucoup plus rapidement tout en limitant le risque d'une mise en péril l'ensemble de l'applicatif.

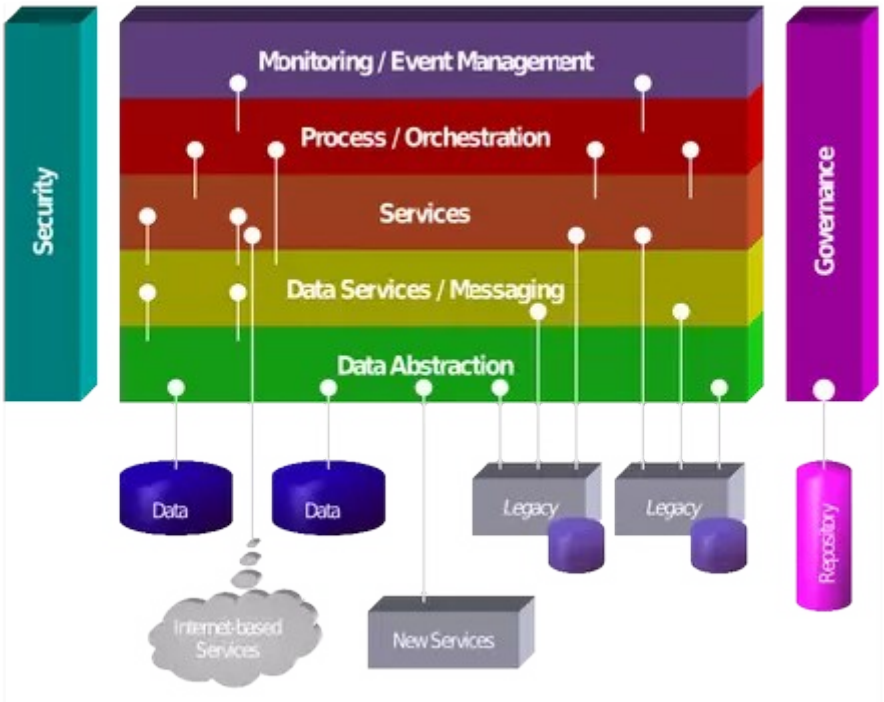
Les Microservices peuvent communiquer entre eux sans état, à l'aide d'interfaces de programmation d'application (API) indépendantes de tout langage.

Le choix technologique dans une architecture Microservices est donc totalement ouvert. Il dépend majoritairement des besoins, de la taille des équipes et de leurs compétences.

Ces architectures d'applications faiblement couplées qui reposent sur des Microservices avec des APIs et des pratiques DevOps constituent la base des applications cloud-native.

Le développement cloud-native est une manière d'accélérer la création des nouvelles applications, d'optimiser les applications existantes, d'harmoniser le développement et d'automatiser la gestion pour l'ensemble des clouds privés, publics et hybrides.

Le schéma ci-dessous représente une vision simplifiée de l'implémentation des Microservices :



Dans le schéma précédent, il est possible de dénombrer sept parties principales :

- Le **Gestionnaire d'évènement (Monitoring)** : cette partie représente une activité de surveillance et de mesure des différents microservices en action. C'est une partie de permettant la supervision globale du système ;
- Le **Planificateur de tâches (Process/Orchestration)** : cette partie permet de planifier l'exécution automatique et périodiques de microservices et ainsi de les ordonnancer en vue de rendre leur exécution cohérente et pertinente vis à vis du système global ;
- Les **Services de données** : cette partie propose une technologie évoluée d'échange de données via un réseau et dont les données échangées sont regroupées dans un Espace Global de Données distribué dans le réseau pour éviter les problématiques liées aux goulots d'étranglements et aux pannes de gestionnaires de données ;
- L'**Abstraction de données** : cette partie est représentée par un modèle mathématique de types de données définissant le comportement d'un microservice en fonction de la sémantique d'une donnée d'un point de vue utilisateur.
- La **Sécurité** : la sécurité globale d'un système utilisant des microservice est appliqué indépendamment à chacun d'entre eux afin de leur assurer la confidentialité des données qu'ils traitent, leur authenticité, leur intégrité et leur disponibilité. Chacune des opérations réalisées par un microservice est aussi tracée et imputée de façon unitaire ;
- La **Gouvernance** : cette partie définit le système d'information global utilisant l'ensemble des microservices. Cette démarche permet de définir la manière dont le système d'information, au travers des microservices, contribue à la création de valeur en précisant le rôle de chaque microservice utilisé.

Avantages

Le 'Time to market'

C'est certainement l'avantage le plus précieux, celui qui permet d'être plus libre et plus rapide dans l'introduction de nouvelles technologies. Plus innovantes, les entreprises gagnent aussi en compétitivité.

Les microservices peuvent être mis à jour, étendus et déployés indépendamment les uns des autres et par conséquent, beaucoup plus rapidement.

L'indépendance fonctionnelle et technique des microservices permet l'autonomie de l'équipe en charge sur l'ensemble des phases du cycle de vie (développement, tests, déploiement, et exploitation), et favorise par conséquent l'agilité et la réactivité.

En outre, il est possible pour les développeurs de faire évoluer les microservices dont ils ont la charge selon des cycles de vie différents et donc d'agir là où c'est important, et notamment :

- Création de valeur ;
- Réactivité aux évolutions du marché ;
- Satisfaction des utilisateurs.

L'agilité technologique

Le choix technologique dans une architecture Microservices est totalement ouvert. Il dépend majoritairement des besoins, de la taille des équipes et de leurs compétences et peut être adapté aux besoins spécifiques de chaque microservice.

En effet, les microservices étant indépendants et interopérables, ils ne sont pas contraints à une uniformité technologique.

Cette flexibilité technologique permet d'introduire des innovations technologiques progressivement, en limitant le risque encouru : le périmètre impacté est réduit à celui du micro-service concerné

Il est par exemple possible d'utiliser un serveur de bases de données non-relationnel plus performant en écriture et plus extensible en fonction d'un besoin particulier, comme les Time Series.

Il sera possible de privilégier une technologie particulière dans chaque microservice pour répondre à un besoin, par exemple, développer des services de calcul scientifique en C, C++ ou en Python pour profiter d'un code efficace en exécution avec des capacités de distribution ou encore de bibliothèques spécifiques, alors qu'un autre service sera développé en C#.

La modernisation facilitée

Passer à une architecture microservices permet de moderniser son application notamment lors du basculement d'un mode OnPremise vers le Cloud ou encore dans le cadre de l'évolution du business model d'une application.

L'approche incrémentale permet de décomposer l'application en fonctionnalités qui sont isolées au sein de microservices. Lorsque l'application est assez bien découpée, elle peut cohabiter avec une architecture microservices, le temps de moderniser le reste de l'application.

L'évolutivité

Dans une application, certaines fonctionnalités sont plus utilisées que d'autres.

À mesure que la demande de certains services augmente, il est possible d'étendre les déploiements sur plusieurs serveurs et infrastructures pour répondre spécifiquement aux besoins.

Si un service pour une raison ou pour une autre, par exemple pour une question de coût, devait être revu, il serait plus simple de reprendre le développement d'un seul de ces services pour l'adapter à la nouvelle plateforme plutôt qu'adapter l'intégralité d'une application monolithique.

En effet, le délai de mise en production des évolutions fonctionnelles est réduit au délai nécessaire à la mise en œuvre des évolutions du ou des micro-services concernés.

Pour les mêmes raisons, la suppression d'un micro-service ou son agrégation avec un autre micro-service sont également des opérations nettement plus simples à mener que pour une application monolithique.

De plus, les déploiements de micro-services en production sont gérés de manière indépendante les uns des autres, ce qui réduit également le coût de mise en production d'une évolution technologique et le risque associé.

Ces éléments favorisent l'évolutivité d'une application constituée de micro-services.

La fiabilité

Lorsqu'ils sont développés correctement, ces services indépendants n'ont aucun impact les uns sur les autres. Cela signifie que, lorsqu'un élément tombe en panne, l'ensemble de l'application ne cesse pas de fonctionner comme c'est le cas avec le modèle monolithique.

La distribution des micro-services en processus systèmes techniquement indépendants permet donc une meilleure continuité de service.

Cette gestion des risques s'accompagne généralement d'un retour sur investissement optimisé.

Inconvénients

Au fur et à mesure de l'évolution d'un système d'information vers un architecture de microservices, les développeurs applicatifs doivent se soucier de la qualité de service en termes de performance, d'évolutivité et de transparence de la localisation ; ces notions n'étant pas forcément en adéquation avec l'état d'esprit des développeurs. Sans cette prise de conscience de la part des développeurs, le système peut rapidement devenir lourd et présenter des ralentissement, voire des interruptions de services.

Les développeurs doivent alors faire face à la complexité supplémentaire de la création d'un système distribué en :

- implémentant le mécanisme de communication inter-services et gérant les pannes partielles ;
- mettant en œuvre des requêtes couvrant plusieurs services (ce process nécessite une coordination minutieuse entre les équipes) ;
- testant les interactions entre les services ;
- utilisant des outils de développement/IDE orientés pour la création de microservices. En effet, certains IDE, spécialisés pour la création d'applications monolithiques, ne fournissent pas de support explicite pour le développement d'applications distribuées.

Suite à leur intégration, il persistera une complexité relative au déploiement. En production, la complexité opérationnelle de déploiement et la gestion d'un système composé de nombreux services différents demanderont une attention particulière.

En outre, relativement à la performance, il faudra tenir compte d'une augmentation de la consommation de mémoire : l'architecture de microservice remplace N instances d'application monolithiques par NxM instances de services. Si chaque service s'exécute dans sa propre JVM (ou équivalent), ce qui est généralement nécessaire pour isoler les instances, il y a alors une surcharge de M fois plus d'exécutions JVM. De plus, si chaque service s'exécute sur sa propre machine virtuelle (par exemple, une instance EC2), comme c'est le cas chez Netflix, la surcharge est encore plus élevée...

Utilisation

La mise en place d’une architecture microservices peut être envisagée de deux manières :

1. En utilisant les services REST : c’est la manière la plus usitée et simple et à mettre en place dans un premier temps. Cependant, cette approche présente des inconvénients. Les services REST sont majoritairement des appels synchrones et, pour certains, ne sont pas rejouables simplement en cas d’erreur (principalement les opérations PUT/POST). La scalabilité, dans ce cas, est fortement conditionnée à la mise en place d’un système de balancing permettant d’équilibrer la charge entre les différents services répliqués. On ajoute donc des points de contention au système avec pour conséquences, des pertes de performances.
2. En utilisant un bus de messages : son rôle étant de transmettre les messages, il doit être le plus simple possible aucune logique ne doit y être intégrée. Le système devient totalement asynchrone et les microservices n’ont aucun lien entre eux. Le système devient plus tolérant aux pannes, si un service tombe le bus conservera le message qui sera alors consommé à la remise en place du service. Dans le cas d’un bus, un système de monitoring doit être mis en place pour observer la consommation des messages et l’état des services en général, le système ne générant pas d’erreur en cas de service indisponible.&))

Synthèse

Le tableau suivant recense les différents modèles d'architecture décrits précédemment en exposant leur(s) avantages et inconvénient(s) vis à vis de leur utilisation au sein d'Astra :

Modèle d'architecture	Description	Avantages	Inconvénients	Quand l'utiliser
Client-Serveur	Une structure d'application distribuée qui répartit les tâches entre les fournisseurs d'un service, appelés serveurs, et les demandeurs du service, appelés clients. Les clients et les serveurs communiquent souvent sur un réseau en utilisant des matériels différents.	Encapsulation du matériel, des logiciels et des fonctionnalités. Combinaison fluide de clients et de serveurs sur différentes plateformes.	Si tous les clients demandent simultanément des données au serveur, celui-ci peut être surchargé. Si le serveur échoue pour une raison quelconque, aucune demande client ne peut être satisfaite.	Lorsque les utilisateurs ont différents appareils. Lorsque vous avez besoin d'encapsuler les fonctionnalités du système.
Pilotage par les événements	Les événements sont produits, transportés et interprétés dans un bus d'événements. Les clients s'abonnent à un groupe d'événements (appelé canal) et agissent à la réception des messages.	Capacité à gérer des millions d'événements en même temps. Capacité à faire communiquer différentes technologies et plateformes avec le même bus d'événements.	Si le bus d'événements tombe en panne, le système ne fonctionne plus du tout. Le bus d'événements peut être surchargé et subir des problèmes de performance.	Lorsque vous avez plusieurs événements en même temps. Lorsque vous devez agir sur un événement en temps réel (exécution synchronisée). Lorsque vous avez des plateformes différentes.

Modèle d’architecture	Description	Avantages	Inconvénients	Quand l’utiliser
Centrée sur les données	Une structure d'application qui utilise les données au lieu du code pour prendre des décisions de comportement.	Il n'est pas nécessaire d'analyser tous les cas dans le code. Les catégories peuvent s'étendre indéfiniment. Des modifications peuvent être apportées sans changer le code.	Peut devenir compliqué à comprendre et à gérer lorsqu'il y a de nombreuses catégories.	Lorsqu’il y a de nombreuses catégories différentes et que vous devez agir différemment pour chacune d'entre elles.
En couches	Les logiciels fonctionnent en couches qui permettent à tous les composants d'être indépendants les uns des autres.	Encapsulation du matériel, des logiciels et des fonctionnalités. Si une couche est modifiée, les autres couches restent les mêmes.	Pour les petites applications, de nombreuses couches créent un problème de performance et sont très difficiles à maintenir.	Uniquement pour les grandes applications.
Modulaire	Parties supplémentaires du logiciel développées pour ajouter une fonctionnalité spécifique non prévue dans le système.	Pas de réécriture du système. Augmentation de la fonctionnalité sans limite.	Les modules se font fréquemment planter les uns les autres et produisent des dysfonctionnements dans le système principal.	Lorsque vous avez une fonctionnalité spécifique que vous devez ajouter sans réécrire le système de base.
Orienté services	Un modèle d'architecture basé sur les services qui permet à un système externe d'utiliser une bibliothèque de fonctionnalités sans accéder aux systèmes internes.	Communication simple : fonctionne à 100 % sur Internet. Normes de sécurité strictes : aucun client ne peut accéder aux systèmes internes.	Les clients ont besoin d'Internet pour utiliser la bibliothèque. Le contrôleur de services web peut être surchargé et subir des problèmes de performance.	Lorsque vous avez de nombreux clients pour un service web. Lorsque vous devez communiquer à distance avec ces clients.

Modèle d’architecture	Description	Avantages	Inconvénients	Quand l’utiliser
microservices	Un modèle d’architecture permettant de diviser une application en fonctionnalités encapsulées au sein de services autonomes. Chacun de ces services est géré et évolue indépendamment des autres services.	• Time-to-market • Agilité technologique • Modernisation facilitée • Evolutivité • Fiabilité	La qualité de services peut se détériorer si les développeurs ne se soucie pas de la performance globale du système d’information lors de la réalisation des microservices.	La mise en place de microservices peut être effectuée en utilisant des services REST ou un bus de messages.

Conclusion

Tel qu'il a été énoncé au sein du *§Existant – Baseline Business Architecture*, le modèle d'architecture actuel d'Astra est basé sur une architecture de type Client-Serveur. Or ce modèle a une particularité qui présente un avantage pour le choix du modèle d'architecture de la nouvelle solution. En effet, une architecture Client-Serveur est une architecture qu'il est aisé de faire évoluer vers un autre modèle d'architecture logicielle.

Ainsi, en prenant en compte les présentations des différents modèles, et en corrélant leur avantages et inconvénients aux différentes exigences et contraintes décrites, il s'avère que le modèle d'architecture qui soit le plus approprié aux besoins d'Astra soit le modèle d'architecture des microservices.

En conclusion, vous pourrez trouver ci-dessous la priorisation des modèles d'architecture préconisés dans un ordre croissant, c'est à dire que plus la priorité est grande, moins le modèle est préconisé :

Priorité	Modèle d'architecture
1	Microservices
2	Orienté services
3	Modulaire
4	En couches
5	Centrée sur les données
6	Client-Serveur
7	Pilotage par les événements


Pstra Recherche