# Projet de mise à niveau des outils de collaboration

# Framework 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 | 31/05/2022 | Création du document | Gérald ATTARD |
| | | | |

## Validation

| N° version | Nom & Qualité | Date & Signature | Commentaires & Réserves |
|---|---|---|---|
| 1.0 | Terry Strasberg<br>CTO | | |
| | | | |

## Tableau des abréviations

| Abr. | Sémantique |
|---|---|
| ADM | Architecture Development method (trad. *méthode de développement d'architecture*) |
| BBA | Baseline Business Architecture (trad. *architecture métier de référence*) |
| SRP | Single Responsability Principle (trad. *principe de responsabilité unique*) |
| TBA | Target Business Architecture (trad. *architecture métier cible*) |
| UML | Uniform Modeling Language (trad. *langage de modélisation uniforme*) |

# Table des matières

# Situation actuelle

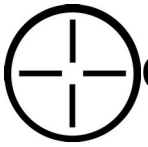
Leader dans le domaine de le recherche médicale depuis plusieurs années, l'IRA (Institut de Recherche Astra) s'est forgée une solide réputation basée sur la qualité de ses recherches médicales et la pertinence des résultats mis en évidence.

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 vue 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 permettre l'identification d'outils, de pratiques, de méthodes pouvant être mis en œuvre pour mener à bien l'implémentation de l'architecture retenue pour le projet.

Ce document, ne décrivant pas précisément l'architecture retenue, fournit des informations connexes.

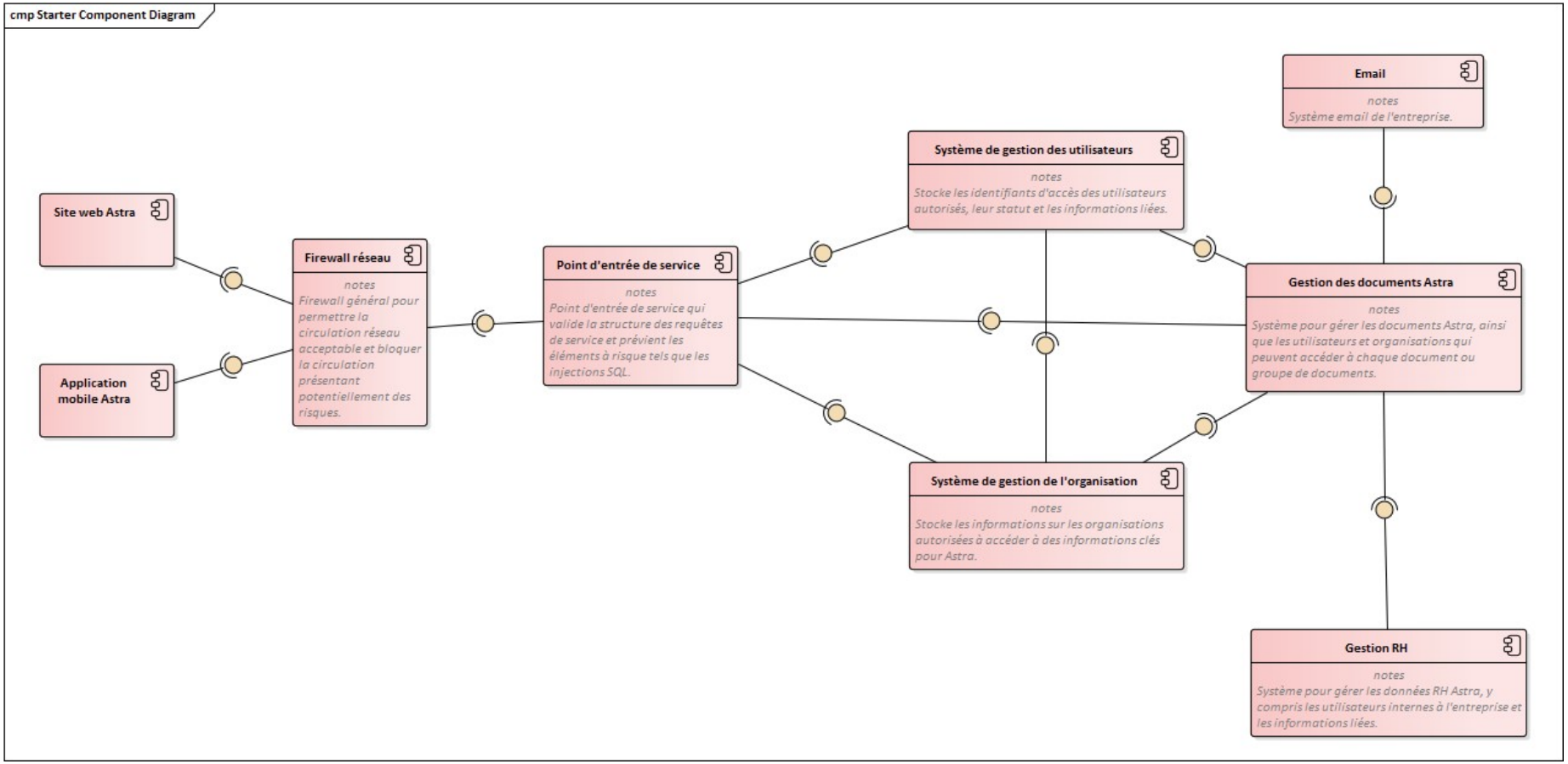
Il se basera sur des ressources documentaires connexes telle que l'étude exploratrice, dans laquelle il sera possible de prendre connaissance de la Baseline Business Architecture, ainsi que de la Target Business Architecture.

Ce document présentera donc les informations suivantes :

- une analyse de l'écart entre le Business Architecture Baseline et le Business Architecture Target, indiquant les composants métiers à ajouter ou à modifier ;
- les bonnes pratiques à appliquer pour mettre en œuvre l'architecture retenue ;
- un cycle de vie relatif à l'implémentation de la solution retenue ;
- un cycle de vie relatif à l'implémentation des nouveaux composants spécifiques à l'extensibilité de l'architecture retenue ;
- les bonnes pratiques concernant l'extensibilité ainsi que la maintenabilité de la solution retenue.

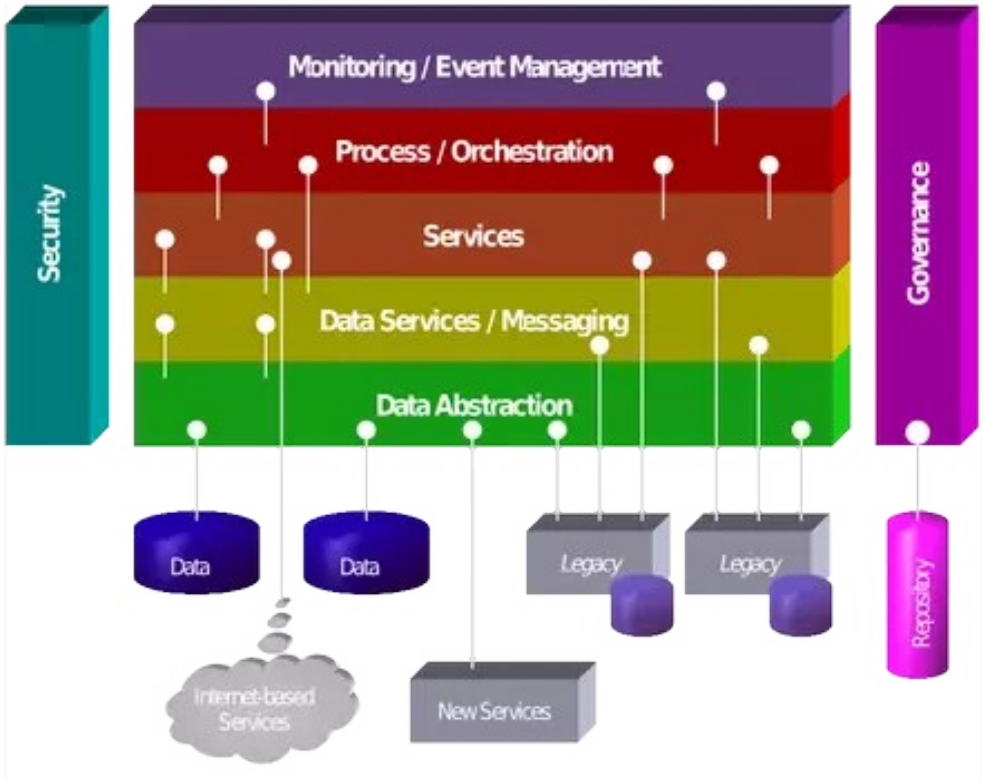
# Baseline Business Architecture

Tel qu'il a été rédigé au sein de l'étude exploratoire associée à ce projet, la *Baseline Business Architecture* se base sur le diagramme UML ci-dessous :

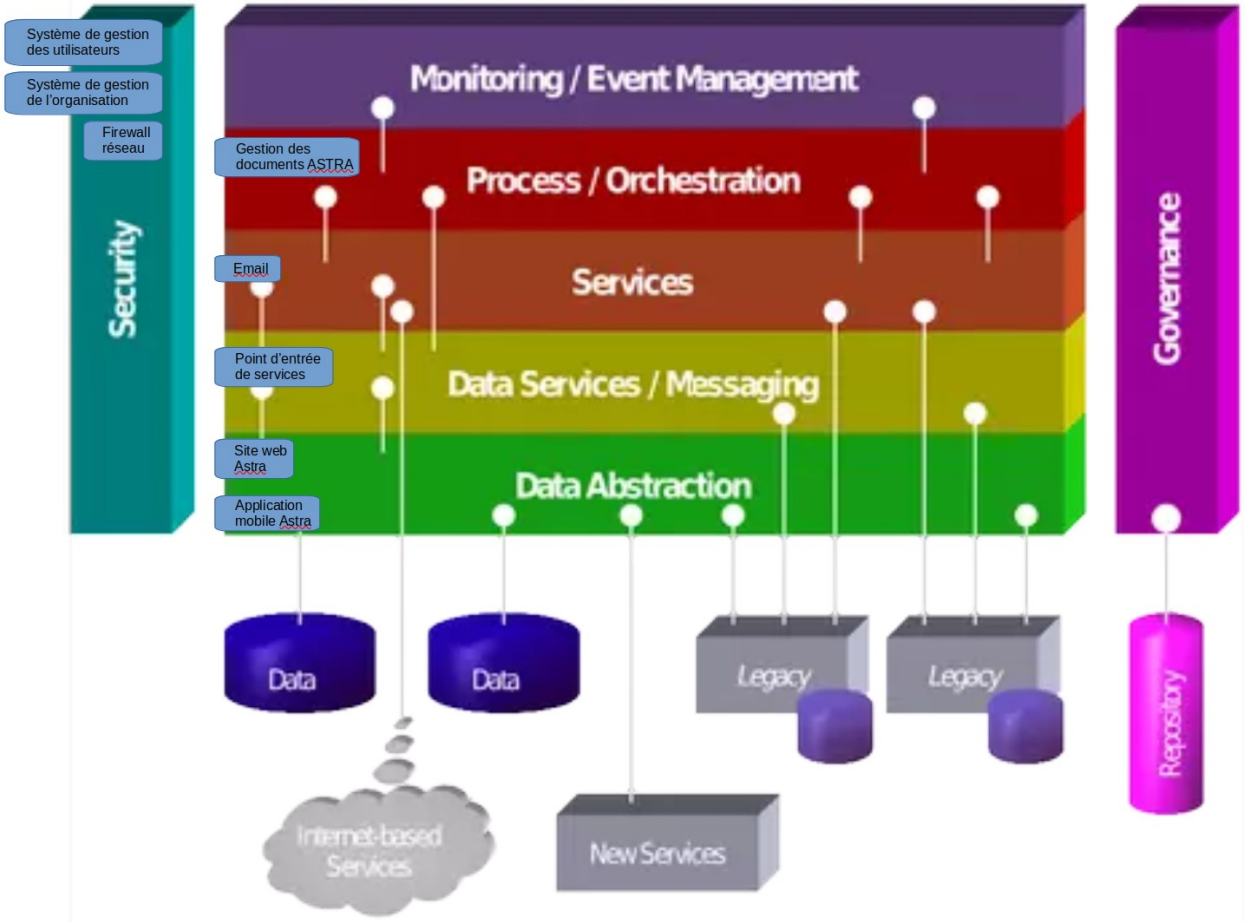


# Target Business Architecture

Tel qu'il a été annoncé au sein de l'étude exploratoire associée à ce projet, la *Target Business Architecture* sera focalisera sur l'utilisation d'un modèle d'architecture des microservices. Ce dernier est défini selon le schéma générique ci-dessous :

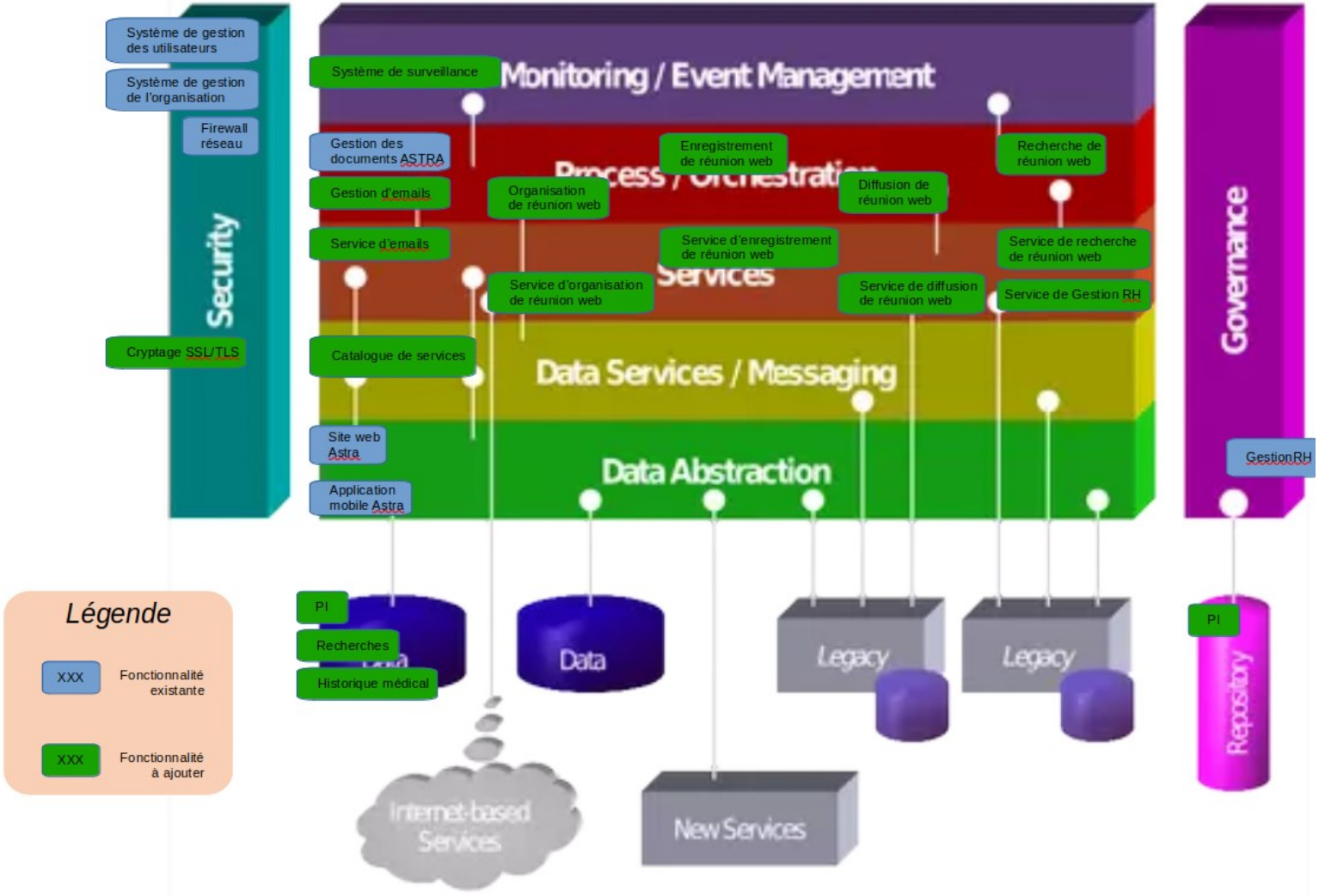


En combinant et superposant les schémas de la *Baseline* et de la *Target Business Architecture*, une solution intermédiaire triviale apparaît :

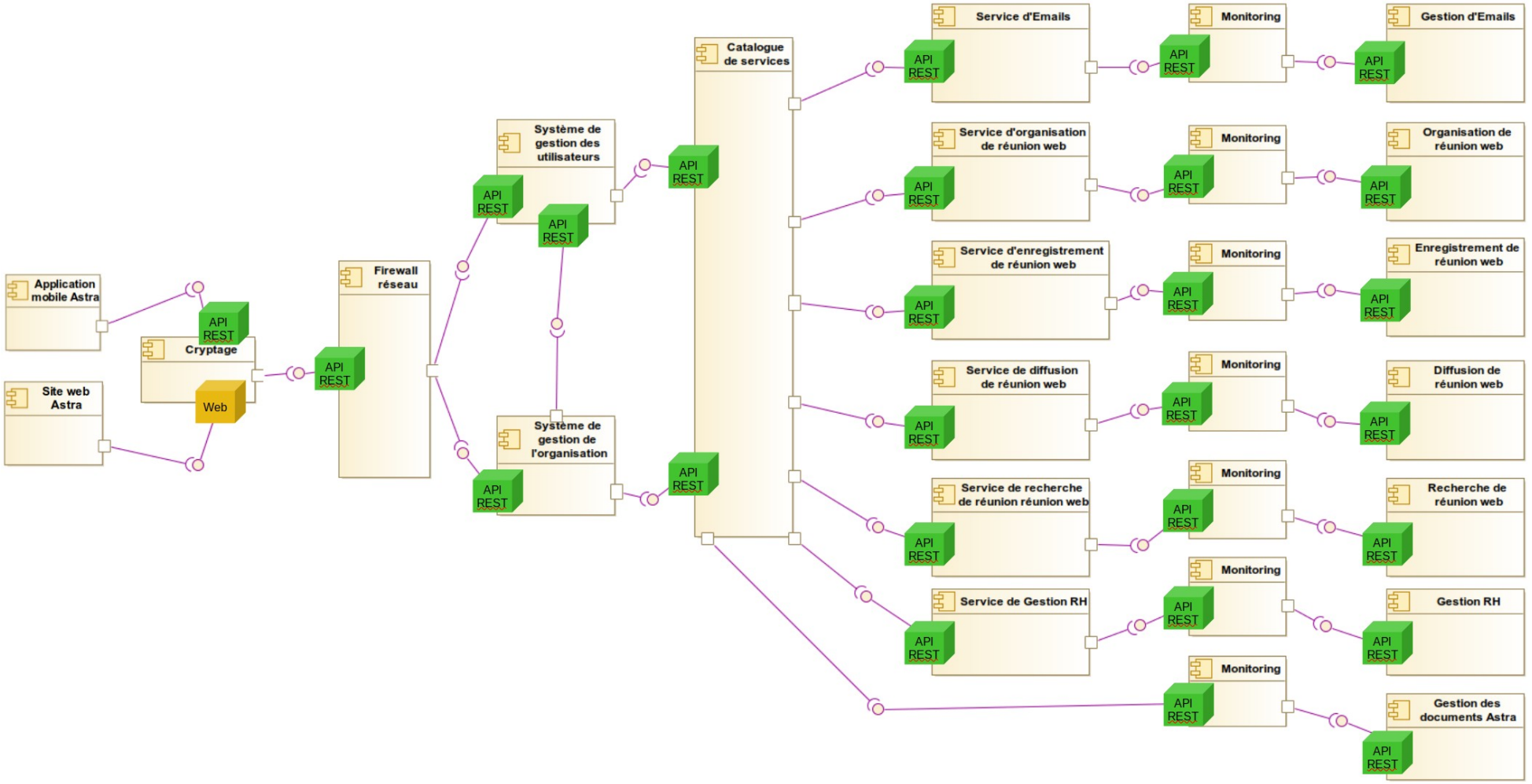



Néanmoins, le schéma ci-dessus reste incomplet et présente plusieurs lacunes dues à l'absence de certains composants permettant la transition de l'architecture Client-Serveur actuelle d'Astra vers une architecture basée sur les microservices.

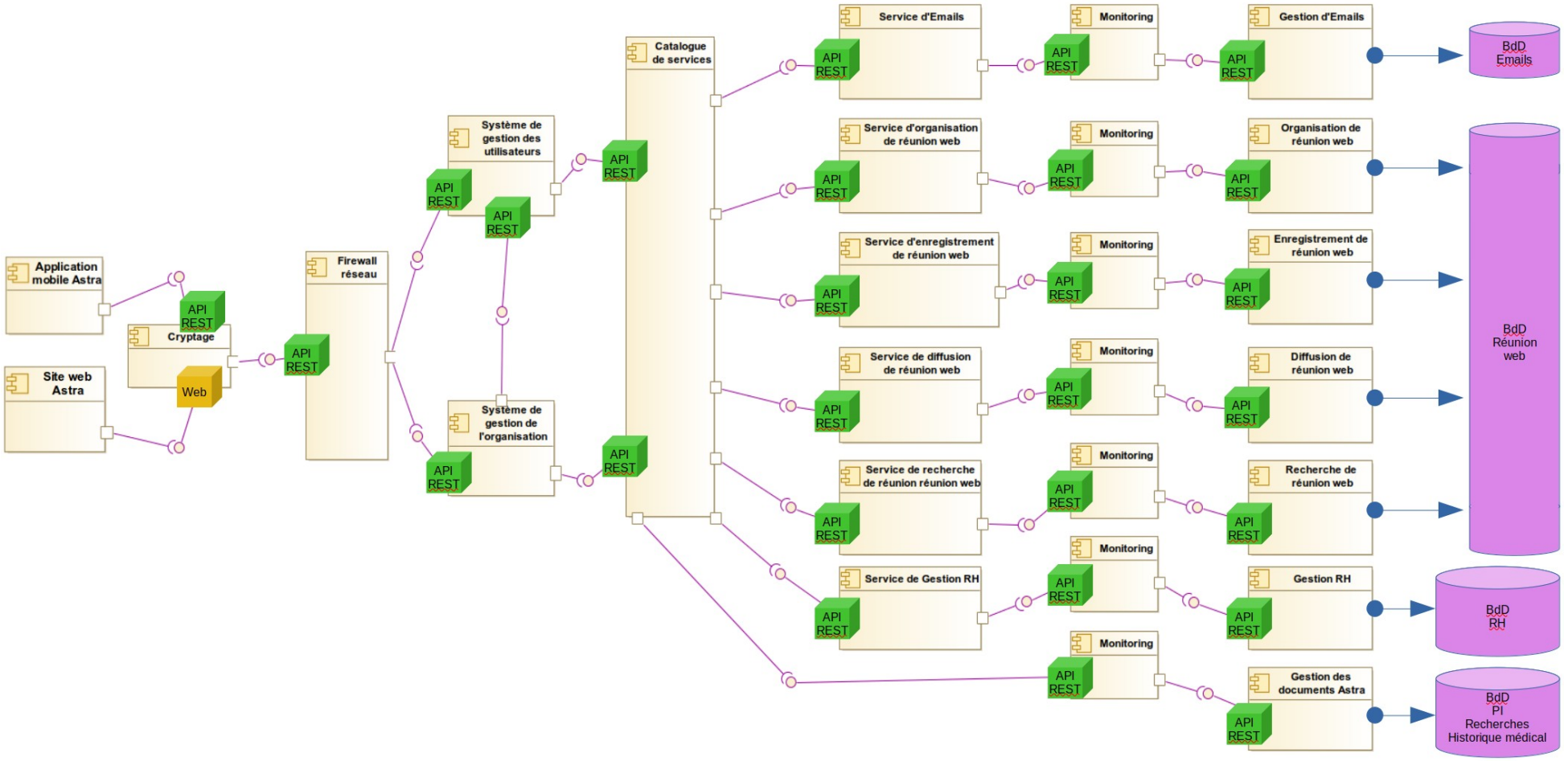
Il s'avère donc nécessaire de rajouter des composants pour effectuer cette transition afin que toutes les fonctionnalités décrites dans le *§Objectif-Target Business Architecture* de l'étude exploratoire soient couvertes :



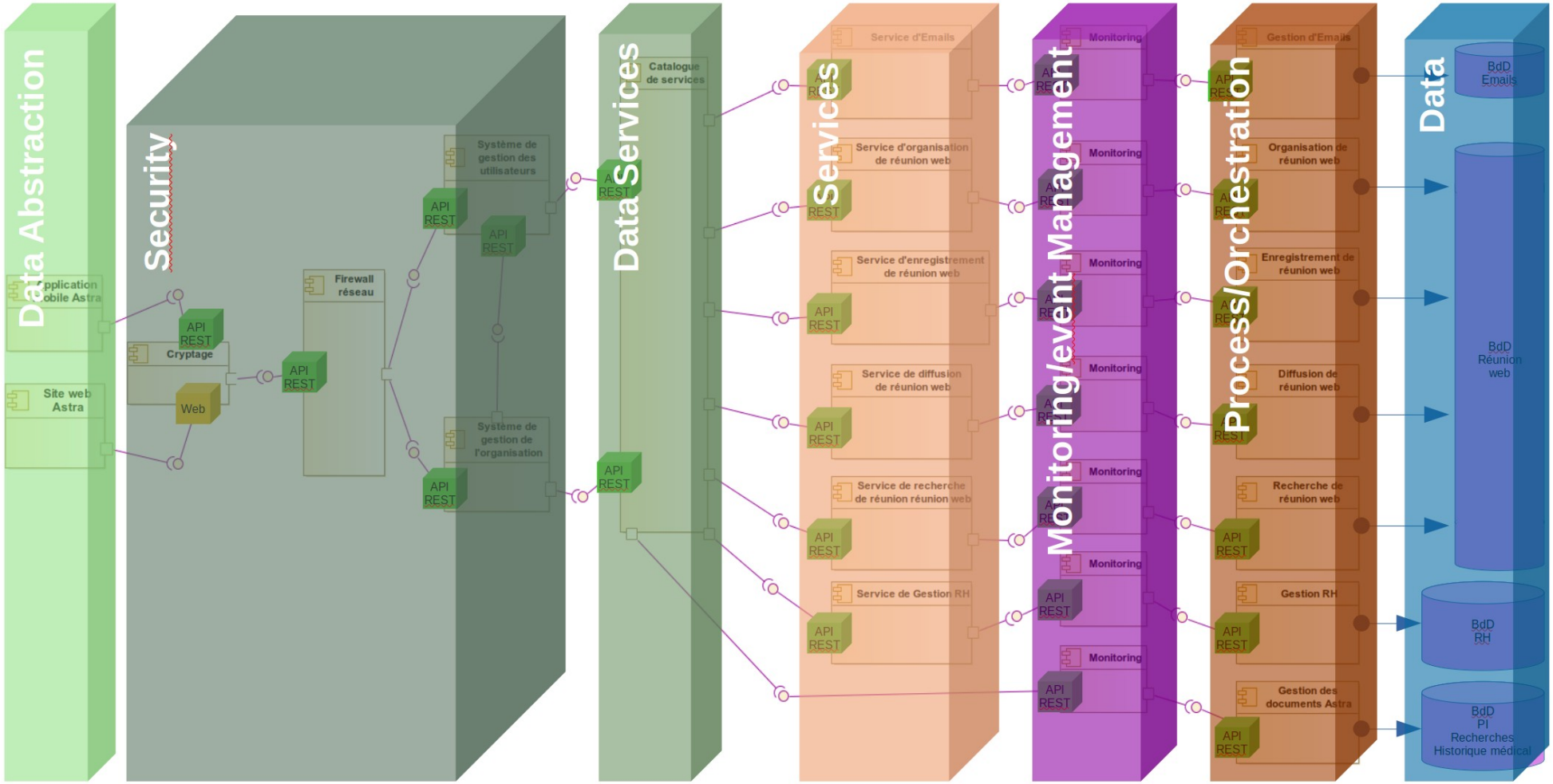
Pour rendre le schéma ci-dessus plus comparable à la représentation initiale de la *Baseline Business Architecture*, il est possible de le transposer en un diagramme de composants, ce qui permettra d'inclure les interfaces de communication de chaque composant présenté :



De plus, il est possible d'y inclure également les sources de données, représentées par des bases de données propres à chaque microservice.



Afin de parfaire l'analogie entre la *Baseline* et la *Target Business Architecture*, il est également possible de mettre en avant les différentes sections exposées lors de la présentation générales des microservices, directement sur le diagramme de composants obtenu :



# Définition des composants

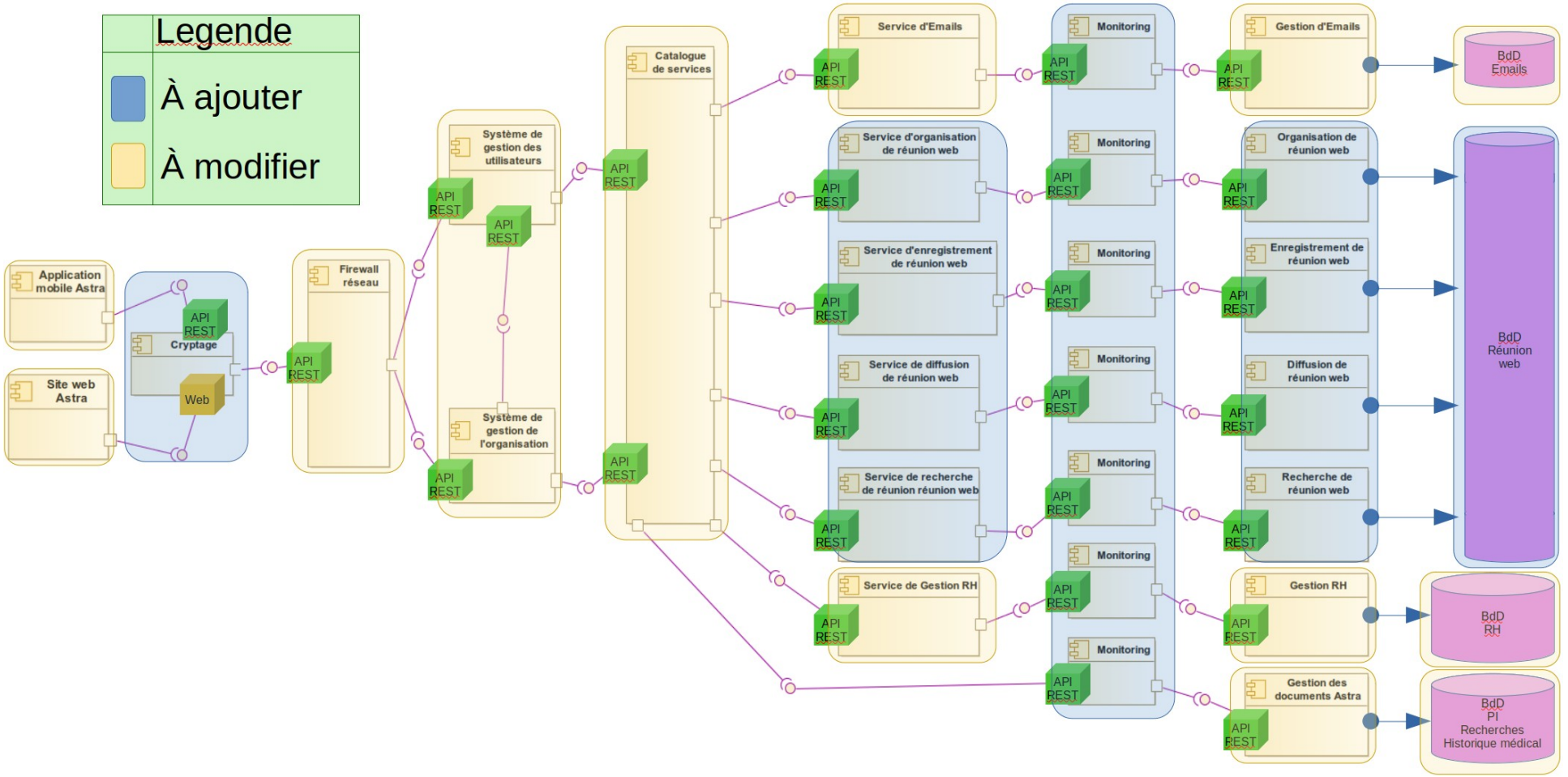
Maintenant que la *BBA* et la *TBA* sont décrites, il convient de définir chaque composant les constituant. Aussi, chaque composant sera notifié des mentions *BBA* ou *TBA*, en fonction de leur appartenance à l'une ou l'autre architecture.

- **Application mobile Astra (*BBA*) :** 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. L'application mobile permet 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 ;
- **Catalogue de services (*TBA) :*** microservice recensant tous les microservices de Service (dans le sens Fonctionnalités offertes) et effectuant la répartition et l'orientation des responsabilités relativement aux requêtes d'entrée ;
- **Cryptage (*TBA) :*** microservice visant à rendre les données illisibles pour tout le monde sauf pour les personnes (ou les microservices) qui possèdent les clefs de déchiffrement. Ici, deux méthodes seront possibles et à utiliser en fonction des besoins, le chiffrement symétrique ou le chiffrement asymétrique ;
- **Diffusion de réunion web (*TBA*) :** microservice basé sur le streaming permettant la diffusion d'un flux audio et/ou vidéo en « direct » ou en différé ;
- **Email (*BBA*) :** service d'email typique pour recevoir et envoyer des emails, pour les utilisateurs internes à Astra. Gère les emails transactionnels envoyés par une API ;
- **Enregistrement de réunion web (*TBA*) :** microservice permettant l'enregistrement de session de diffusion de réunion selon le microservice de diffusion de réunion web ;
- **Firewall réseau (*BBA*) :** firewall général du réseau configuré pour protéger les systèmes Astra de la circulation réseau inattendue ou non planifiée. Fournit l'accès port 80 aux systèmes et services exposés alors que les systèmes internes peuvent utiliser différents ports HTTP pour la protection des données ;
- **Gestion des documents Astra (*BBA*) :** microservice gèrant les documents Astra avec des 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 ;
- **Gestion d'Emails (*TBA*) :** microservice spécialisé dans la gestion des emails reçus basé sur des processus de dissociation des méta-données (en-tête de message) et des données (corps de message) reçues au sein de chaque email ;
- **Gestion RH (*BBA)* :** système pour gérer les utilisateurs, salariés et prestataires internes à Astra, inclut le rôle, le département et les permissions d'accès de l'utilisateur ;
- **Monitoring (*TBA*) :** microservice spécifique à chaque microservice auquel il est associé pour surveiller et définir une mesure d'activité idoine ;
- **Organisation de réunion Web (*TBA*) :** microservice permettant la planification d'une future réunion web ou initiant la réalisation d'une réunion web immédiate ;
- **Point d'entrée de service (*BBA*) :** dispositif vérifiant que les utilisateurs accèdent uniquement aux services auxquels ils ont accès ;

- **Recherche de réunion web (*TBA*) :** microservice permettant de trouver les informations relatives à une réunion web ayant déjà été effectuée ou ayant été planifiée ultérieurement ;
- **Service de diffusion de réunion web (*TBA*) :** microservice gérant les méta-données associées à la diffusion d'une réunion web spécifique ;**Service d'Emails (*TBA)* :** Service d'email typique pour recevoir et envoyer des emails, pour les utilisateurs internes à Astra. Gère les emails transactionnels envoyés par une API ;
- **Service d'enregistrement de réunion web (*TBA*) :** microservice gérant les méta-données associées à l'enregistrement d'une réunion web spécifique ;
- **Service d'organisation de réunion web *(TBA)* :** microservice gérant les méta-données associées à l'organisation d'une réunion web spécifique ;
- **Service de Gestion RH (*TBA*) :** microservice gérant les méta-données associées au système de gestion RH ;
- **Site Web Astra (*BBA*) :** 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. Le site web est construit comme une application web réactive permettant l'accès depuis une variété d'appareils et de tailles d'écran.
- **Système de gestion des utilisateurs (*BBA*) :** système pour gérant les utilisateurs ayant la permission d'accéder aux services et à d'autres systèmes internes, dont le rôle, l'authentification, et les capacités liées des utilisateurs ;
- **Système de gestion de l'organisation (*BBA*) :** système gérant les organisations ayant accès aux données et services Astra. Les utilisateurs doivent appartenir à une organisation autorisée. Certains services permettent à tout utilisateur d'une organisation d'accéder à des données et documents limités.

# Différence de composants

Bien que cela ait été effectué indirectement dans la définition de chaque composant au paragraphe précédent, il convient de réaliser à présent l'identification précise des composants à modifier ou à ajouter lors de la migration du *BBA* vers le *TBA* de microservices :



Relativement au schéma ci-dessus, il est possible de le synthétiser en fonction de la modification (*MOD*) et de l'ajout (*ADD*) des composants recensés :

| Composant | Emplacement | Opération |
|---|---|---|
| Application mobile Astra | BBA | MOD |
| Catalogue de services | TBA | MOD |
| Cryptage | TBA | ADD |
| Diffusion de réunion web | TBA | ADD |
| Email | BBA | MOD |
| Enregistrement de réunion web | TBA | ADD |
| Firewall réseau | BBA | MOD |
| Gestion des documents Astra | BBA | MOD |
| Gestion d'Emails | TBA | MOD |
| Gestion RH | BBA | MOD |
| Monitoring | TBA | ADD |
| Organisation de réunion Web | TBA | ADD |
| Point d'entrée de service | BBA | MOD |
| Recherche de réunion web | TBA | ADD |
| Service de diffusion de réunion web | TBA | ADD |
| Service d'Emails | TBA | MOD |
| Service d'enregistrement de réunion web | TBA | ADD |
| Service d'organisation de réunion web | TBA | ADD |
| Service de Gestion RH | TBA | MOD |
| Site Web Astra | BBA | MOD |
| Système de gestion des utilisateurs | BBA | MOD |
| Système de gestion de l'organisation | BBA | MOD |

# Interactions entre composant

L'identification précise des composants étant réalisée, il convient à présent d'analyser les interactions entre ces composants. Pour cela, cette étude se basera sur les diagrammes d'interaction UML. Le but de ces diagrammes est de capturer l'aspect dynamique d'une système, c'est à dire l'instantané du système en cours d'exécution à un moment particulier.
Ainsi, les diagrammes utilisés seront :

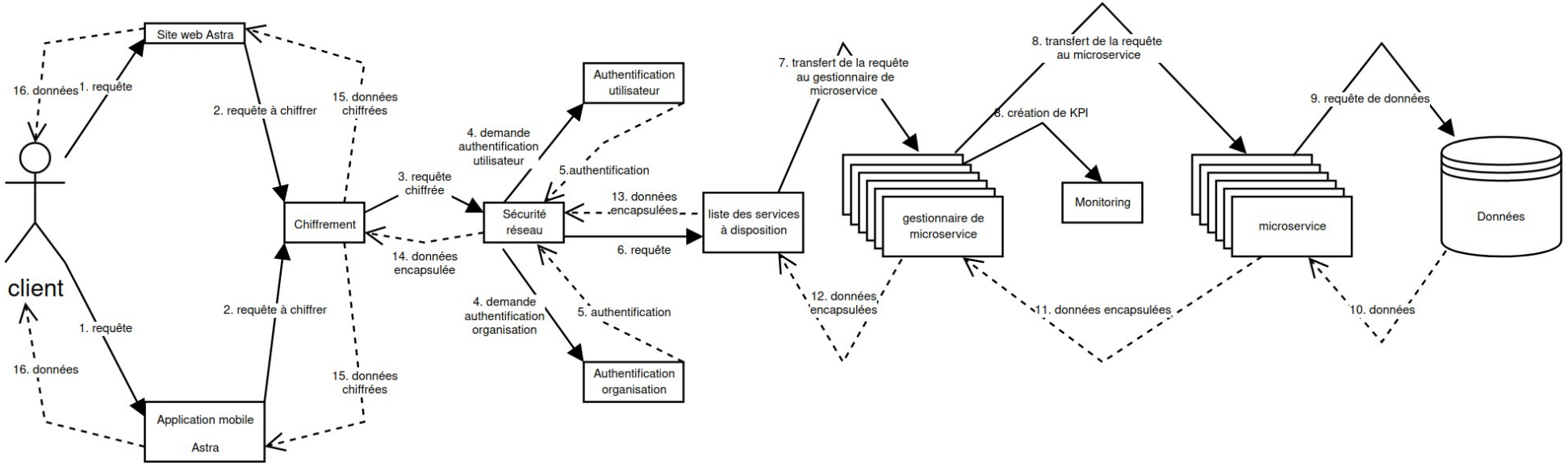
- le diagramme de collaboration pour décrire l'organisation structurelle des objets participant à l'interaction ;
- le diagramme de séquence pour capturer l'ordre des messages circulant d'un composant à un autre.

Ces deux types de diagramme sont utilisés pour comprendre :

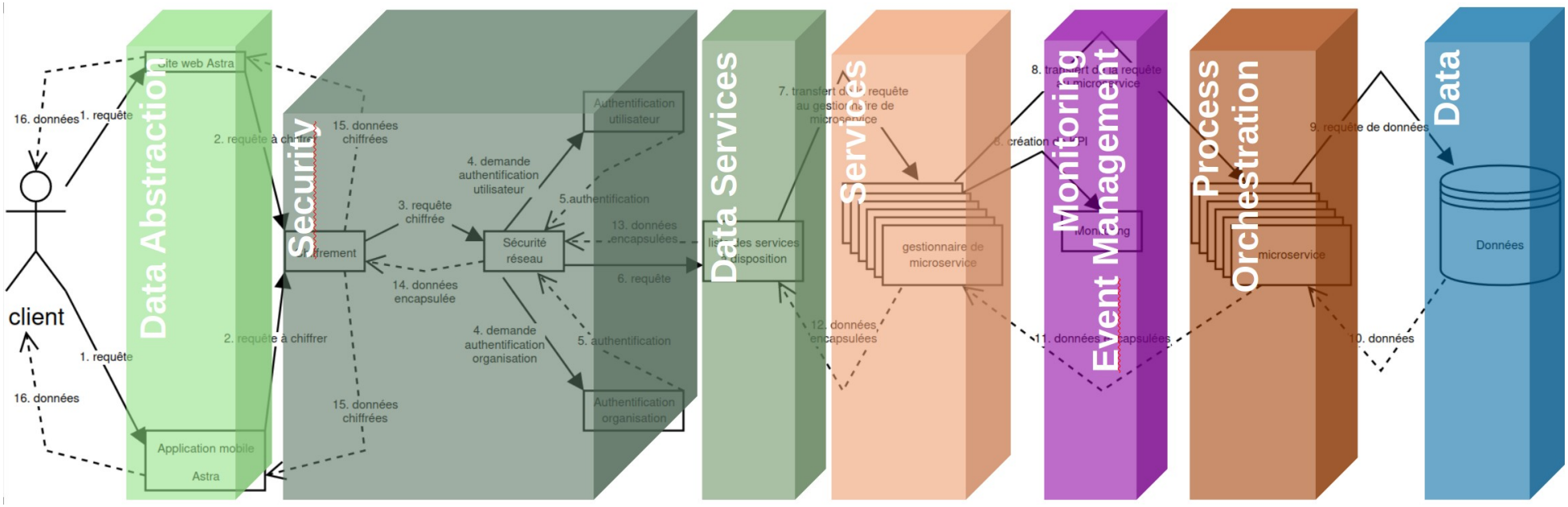
- le flux de message ou séquence du flux de contrôle d'un objet à un autre ;
- l'organisation structurelle et visuelle d'un système.

# Diagramme de collaboration

Le diagramme de collaboration voit son utilité dans la définition des éléments utiles pour obtenir un résultat en spécifiant les RÔLES et les INTERACTIONS des composants précédemment identifiés. Ainsi, le schéma ci-dessous représente ces rôles ainsi que les messages de retours propres à chacun :



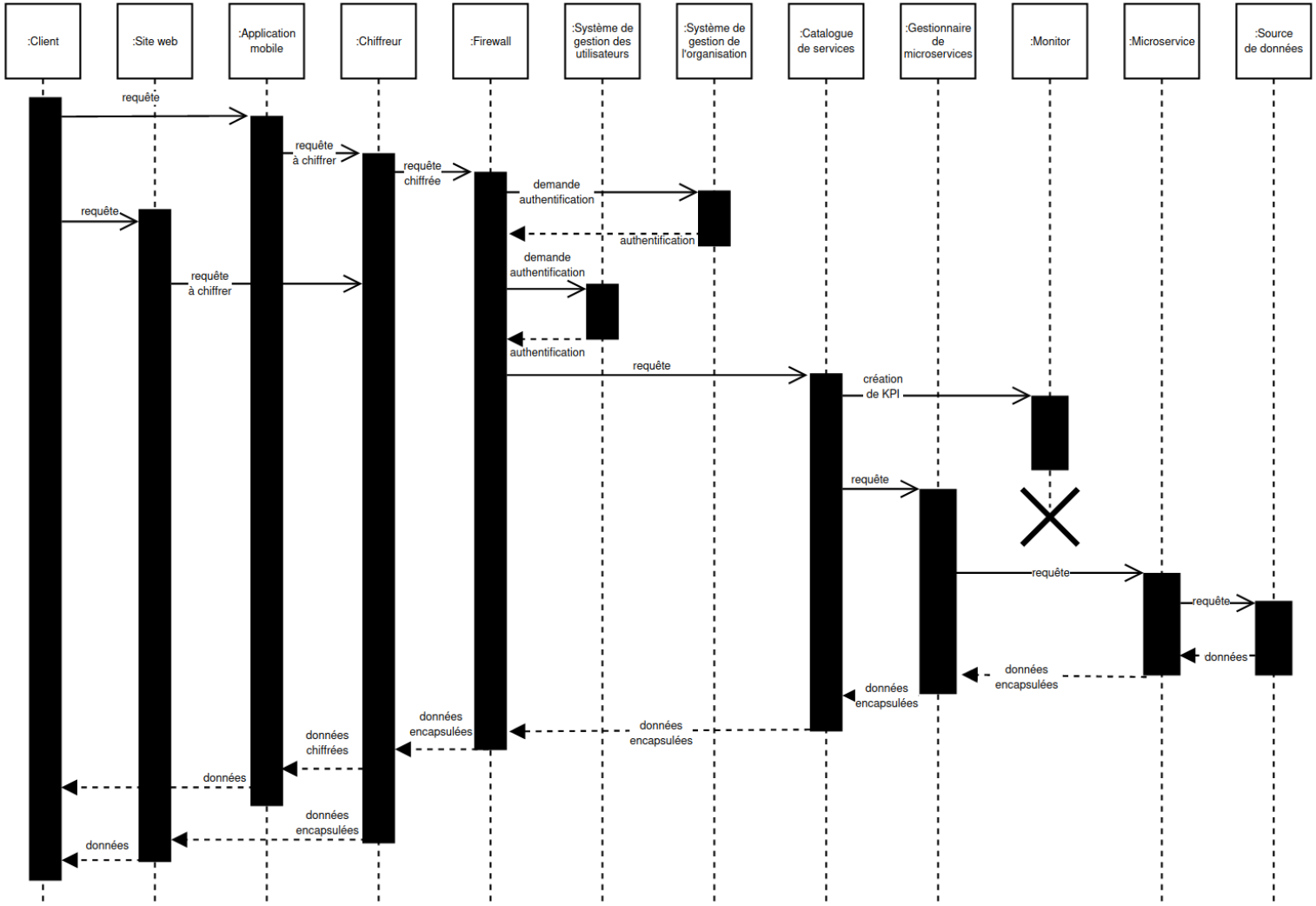
Afin de s’assurer de la cohérence du précédent diagramme de collaboration, il est toujours possible d’y superposer le schéma de présentation générale des microservices afin de corréler les composants à leurs rôles respectifs :




La superposition étant cohérente, les rôles identifiés sont donc en corrélation avec leur composant associé.

# Diagramme de séquence

En se basant sur le diagramme de collaboration élaboré précédemment, il est alors possible d'en déduire un diagramme de séquence. Ce dernier se concentre sur les messages que les composants échangent entre eux pour exercer une fonction avant la fin de la ligne de vie.



# Cycle de vie

Dans les paragraphes précédents , ce document a identifié :

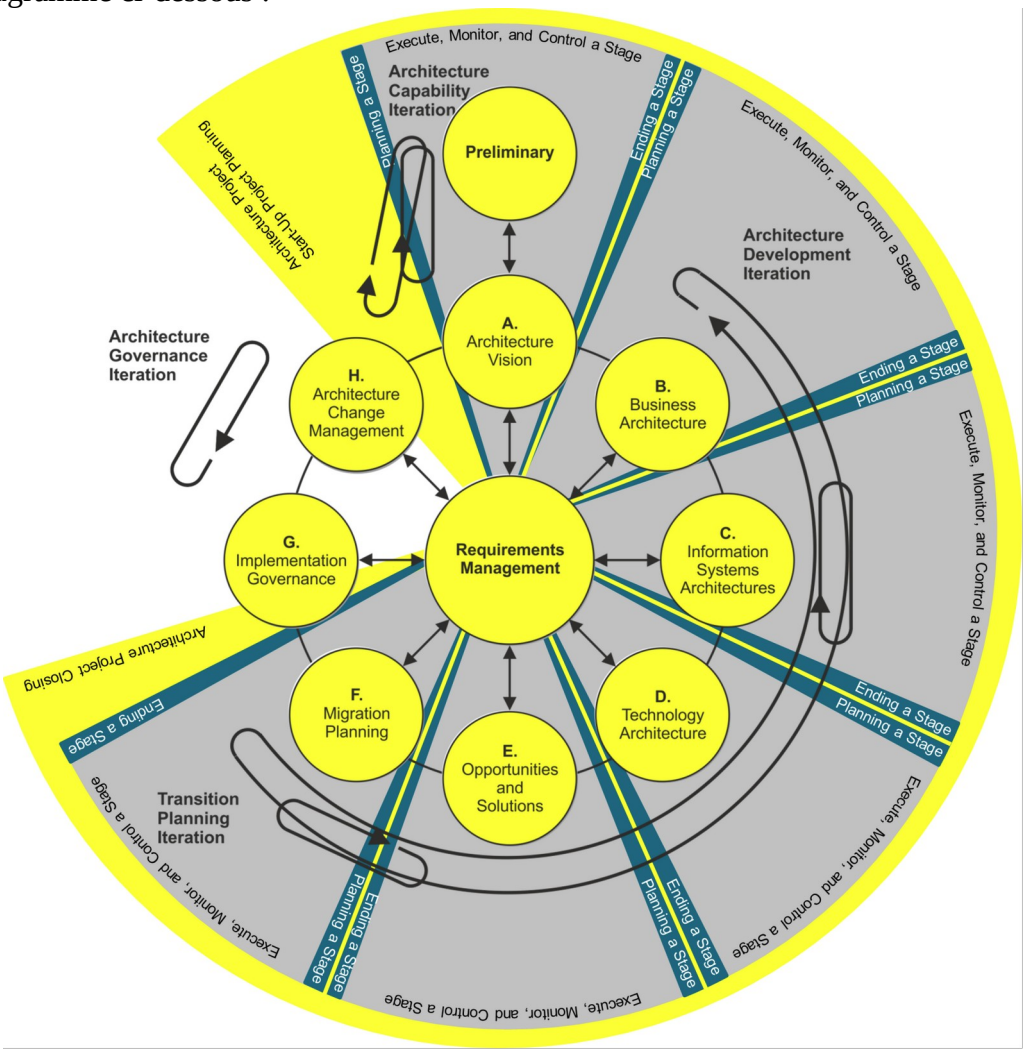
- les différents composants du modèle d'architecture préconisé, à savoir les microservices ;
- les diverses interactions entre ces composants.

En complément, il reste encore à identifier les cycles de vie relatifs à :

- l'implémentation de cette nouvelle l'architecture ;
- l'implémentation des nouveaux composants de cette nouvelle architecture.

# Implémentation de l'architecture

Tel qu'il est mentionné dans le TOGAF®,l'implémentation d'un projet d'architecture englobe des activités de gestion qui peuvent être regroupées et séquencées en différentes phases représentées dans le diagramme ci-dessous :



Le schéma précédent représente le cycle de vie d’un projet d’architecture constitué des étapes suivantes :

- **début du projet d’architecture :** étape représentant le démarrage du projet d'architecture. Elle soulève des questions liées à la nomination de l'exécutif et du chef de projet, ainsi que de l'équipe de gestion de projet, à la capture des leçons précédentes, à la sélection de l'approche du projet, à l'assemblage du descriptif du projet et à la planification de l'étape suivante ;

- **planification de projet d'architecture :** phase décrivant les activités de cadrage et de planification du projet en fonction des méthodes de gestion de projet ;

- **planification d'étape :** phase divisant un projet d'architecture en étapes et planifiant chaque étape en alignement avec un plan de projet global ;

- **exécution, surveillance et contrôle d’étape :** phase précisant comment surveiller et contrôler les activités de développement d'architecture des phases ADM B à F (cf. TOGAF®) à l'aide de méthodes de gestion de projet ;

- **finalisation d’étape :** phase expliquant les étapes clés effectuées à la fin d'une étape ;

- **clôture du projet d'architecture :** phase fournissant les livrables d'architecture pour la mise en œuvre. Elle détaille les activités de clôture du projet et le transfert des produits du projet d'architecture aux programmes et projets de mise en œuvre, ainsi que l'approche pour documenter les leçons apprises.

# Implémentation des nouveaux composants

Comme il est stipulé dans le *§Différence de composants*, les nouveaux composants sont ceux énumérés ci-dessous :

- microservice de chiffrement ;
- microservice de gestion de l’organisation de réunion web ;
- microservice de gestion de l’enregistrement de réunion web ;
- microservice de gestion de la diffusion de réunion web ;
- microservice de gestion de la recherche de réunion web ;
- microservice d’organisation de réunion web ;
- microservice d’enregistrement de réunion web ;
- microservice de diffusion de réunion web ;
- microservice de recherche de réunion web ;
- microservice de monitoring.

Ainsi, par ordre de priorité d'implémentation, il serait souhaitable de développer les microservices précédents en suivant la séquence de tâches ci-dessous :

- API REST du microservice de gestion de l'organisation d'une réunion web ;
- Microservice de gestion de l'organisation d'une réunion web ;
- API REST du microservice d'organisation d'une réunion web ;
- Microservice d'organisation d'une réunion web ;
- API REST du microservice de monitoring relatif à la gestion de l'organisation d'une réunion web ;
- Microservice de monitoring des microservices relatifs à l'organisation d'une réunion web ;
- API REST du microservice de gestion de la diffusion d'une réunion web ;
- Microservice de gestion de diffusion d'une réunion web ;
- API REST du microservice de diffusion d'une réunion web ;
- Microservice de diffusion d'une réunion web ;
- API REST du microservice de monitoring des microservices relatifs à la diffusion d'une réunion web ;
- Microservice de monitoring des microservices relatifs à la diffusion d'une réunion web ;
- API REST du microservice de gestion de la recherche d'une réunion web ;
- Microservice de gestion de recherche d'une réunion web ;
- API REST du microservice de recherche d'une réunion web ;
- Microservice de recherche d'une réunion web ;
- API REST du microservice de monitoring des microservices relatifs à la recherche d'une réunion web ;
- Microservice de monitoring des microservices relatifs à la recherche d'une réunion web ;
- API REST du microservice de gestion d'enregistrement de réunion web ;
- Microservice de gestion d'enregistrement de réunion web ;
- API REST du microservice d'enregistrement de réunion web ;
- Microservice d'enregistrement de réunion web ;
- API REST du microservice de monitoring des microservices relatifs à l'enregistrement de réunion web ;
- API REST du microservice de chiffrement ;
- Microservice de chiffrement.

Les points ci-dessus sont les tâches principales à réaliser pour implémenter les nouveaux composants relatifs à la migration vers le nouveau modèle d'architecture. Ces étapes ne sont pas exhaustives et devront être complétées par les *user stories* rédigées au fur et à mesure de l'avancée du projet.

Il est cependant possible d'en extraire une certaine séquence pour la réalisation d'un microservice spécifique :

- écriture de l'API REST du microservice de gestion :
- écriture du microservice de gestion ;
- écriture de l'API REST du microservice ;
- écriture du microservice ;
- écriture de l'API REST du microservice de monitoring ;
- écriture du microservice de monitoring.

# Technologies

# API REST ou RESTful

Une API REST est une interface fréquemment utilisée pour les microservices, dont l’acronyme signifie REpresentational State Transfer.

Ainsi, REST est un style d'architecture logicielle définissant un ensemble de contraintes à utiliser pour créer des services web.

Les services web conformes au style d'architecture REST, aussi appelés services web RESTful, établissent une interopérabilité entre les ordinateurs sur Internet.

Les services web REST permettent aux systèmes effectuant des requêtes de manipuler des ressources web via leurs représentations textuelles à travers un ensemble d'opérations uniformes et prédéfinies sans état.

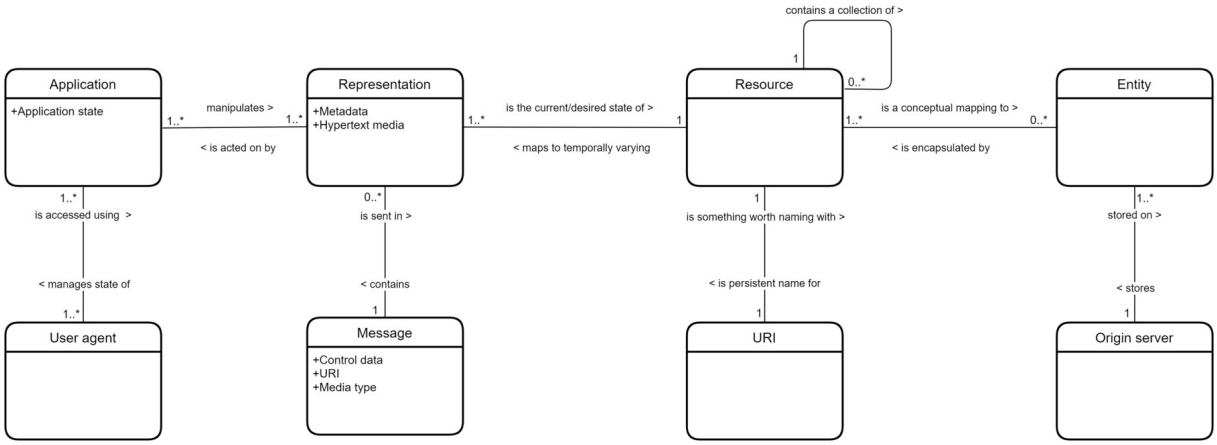
L’utilisation d’une API REST consiste à interroger un serveur tiers en utilisant les mêmes méthodes que ce que propose l’affichage de page web ou des formulaires inclus dans ces mêmes pages web. Ainsi, on va alors interroger un serveur à partir d’une URL communiquée par l’éditeur de l’API.

Cette interrogation va se faire suivant différentes méthodes : GET, POST, PUT et DELETE.

Point important, il ne s’agit pas simplement de faire des requêtes vers une API, il faut aussi savoir ce qu’elles provoquent. En règle générale, une requête attend en retour la réception d’une réponse. Cette réponse est une suite de données renvoyées à l’expéditeur avec le résultat de la requête.

Deux grands formats de retour d’information sont utilisés : le JSON et le XML.

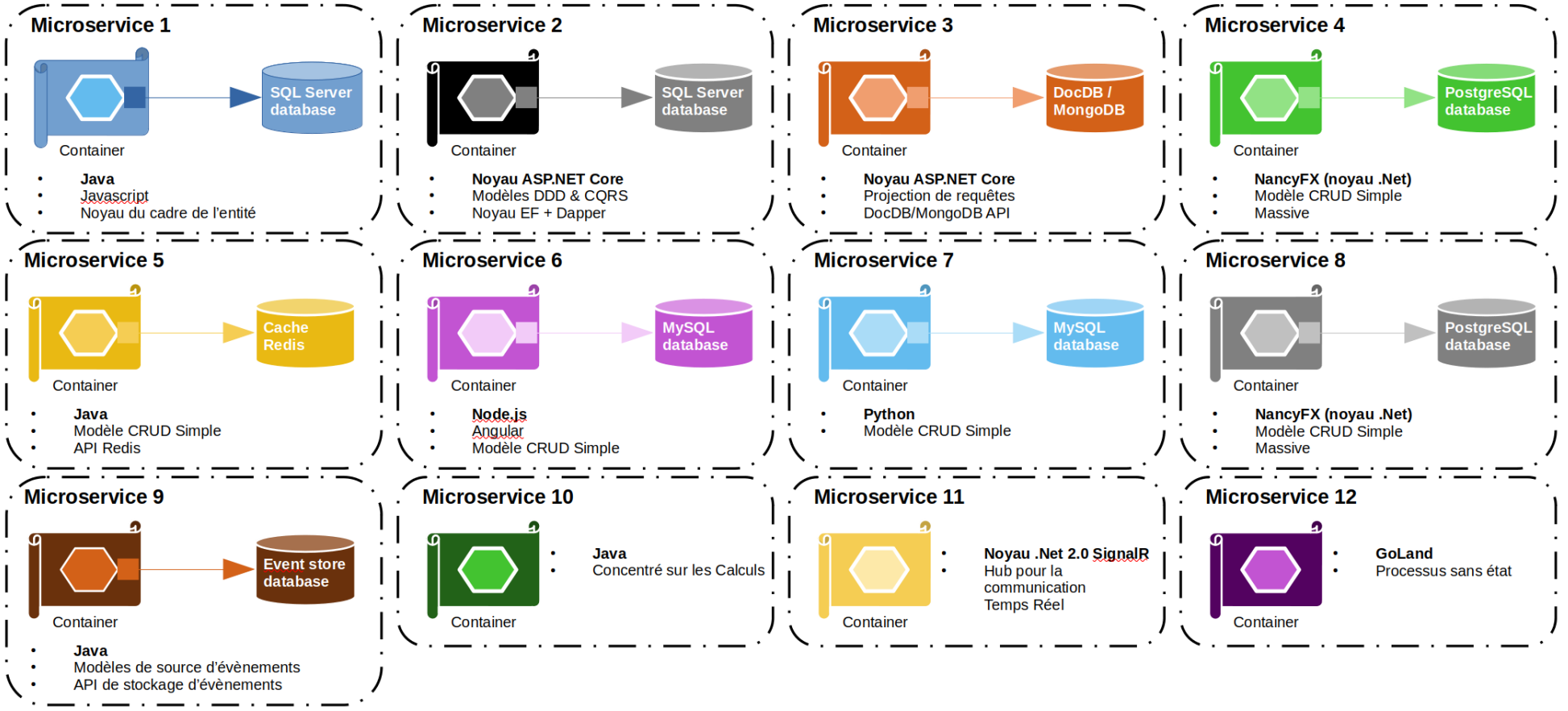
Le diagramme ci-dessous représente l’organisation type d’une API REST :




Cette topologie d’application est optimale pour les microservices, de par leur indépendance les uns vis à vis des autres.

# Les microservices, un monde polyglotte

Peu importe le modèle ou le style d'architecture adopté, il y a un point commun à tous : aucune technologie particulière, ne convient à toutes les situations. La figure ci-dessous montre certaines approches et technologies (mais pas dans un ordre particulier) qui pourraient être utilisées dans différents microservices, ainsi il est possible d'en conclure que les microservices ne sont pas affiliés à une seule et même technologie :



# Bonnes pratiques

Les solutions présentées dans les paragraphes précédents doivent s’accompagner des méthodes d’application idoines quant à leur mise en œuvre.

Ainsi, ce paragraphe fournira les bonnes pratiques à appliquer pour migrer vers un modèle d’architecture en microservices, ainsi que des conseils relatifs à la maintenabilité et l’extensibilité d’une telle architecture.

## Relatives à l’architecture des microservices

L'un des défis liés à l'utilisation de cette approche est de décider quand il est adapté de l'utiliser.

Lors du développement de la première version d'une application, il est difficilement concevable d’appréhender les problèmes que cette approche résout.

De plus, l'utilisation d'une architecture distribuée élaborée ralentira forcément le développement. Cela peut être un problème majeur pour les entreprises dont le plus grand défi est souvent de savoir comment faire évoluer rapidement le modèle commercial et l'application qui l'accompagne.

L'utilisation de fractionnements des fonctionnalités en microservices peut rendre beaucoup plus difficile les itérations de développement et présente des avantages indéniables en terme de robustesse, de fiabilité, de maintenabilité et d’extensibilité.

Néanmoins, lorsque le défi est de savoir comment évoluer et que le choix d’utilisation d’une architecture en microservice est acté, il est alors nécessaire d’utiliser la décomposition fonctionnelle. En effet, les dépendances enchevêtrées peuvent rendre difficile la décomposition de votre système d’information en un ensemble de microservices.

Ainsi, après le choix d’utilisation de ce modèle d’architecture, il est indispensable de décider comment partitionner le système en microservices. Pour faire cela, il existe un certain nombre de stratégies pouvant aider à réaliser ce processus de décomposition :

- décomposer par capacité métier et définir les services correspondant aux capacités métier ;
- décomposer par sous-domaine de conception axée sur le domaine métier ;
- décomposer par verbe ou cas d'utilisation et définissez les services qui sont responsables d'actions particulières comme, par exemple, un service d'expédition responsable de l'expédition des commandes complètes ;
- décomposer par noms ou ressources en définissant un service responsable de toutes les opérations sur les entités/ressources d'un type donné, tel qu’un service de compte qui est responsable de la gestion des comptes d'utilisateurs.

Idéalement, chaque service ne se verra confier qu’un petit ensemble de responsabilités, on parle alors de la conception de classes en utilisant le principe de responsabilité unique (SRP).

Le SRP définit une responsabilité d'une classe comme une raison de changer et stipule qu'une classe ne devrait avoir qu'une seule raison de changer. Il est aisé d'appliquer également le SRP à un niveau plus élevé, tel que la conception de services.

Une autre analogie qui aide à la conception de services est la conception des utilitaires Unix, qui est système d’exploitation fournissant un grand nombre d'utilitaires tels que grep, cat et find. Chaque utilitaire fait exactement une chose, souvent exceptionnellement bien, et est destiné à être combiné avec d'autres utilitaires à l'aide d'un script shell pour effectuer des tâches complexes.

Ainsi, une fois que l’architecture en microservices a été décidée, mise en place et est en cours d’utilisation, il reste encore deux domaines à prendre en compte pour englober cette approche, concernant les bonnes pratiques relatives à :

- la maintenabilité d’une architecture de microservices ;
- l’extensibilité d’une telle architecture.

# Relatives à la maintenabilité

La maintenabilité est une caractéristique précisant la facilité et la rapidité avec lesquelles un système peut être remis en un état de fonctionnement total avec une fiabilité correspondant à son âge.

La rapidité de remise en état d’un système peut être mesurée par la durée active du dépannage dont les temps morts, non imputables à la conception du système, ne seront pas comptabilisés. Ces derniers peuvent être :les délais de réponse des dépanneurs, les durées d’attente des pièces de rechange ou les temps passés à la rédaction des pièces administratives, car ces temps dépendent de l’organisation et de l’efficacité du service de maintenance et non de la conception du système.

Pour rendre le dépannage plus facile et plus rapide, on devra prévoir, dès la conception, les moyens pour faciliter :

- le diagnostic des pannes existantes et de celles risquant de survenir rapidement (défaillances par dégradation) ;
- l’accès aux modules à remplacer, en prévoyant le cas échéant leur débranchement, voire leur remplacement ;
- le contrôle de la validité de l’action de maintenance.

Plus spécifiquement, les microservices permettent une grande maintenabilité dans les grands systèmes complexes, en permettant de créer des applications basées sur de nombreux services qui peuvent être déployés indépendamment, chacun ayant des cycles de vie granulaires et autonomes.

# Relatives à l’extensibilité

L’extensibilité, autrement désigné sous les termes *capacité à monter en charge* ou *capacité à passer à l'échelle,* est issu de l'anglais *scalability ;* il est d’ailleurs courant d’utiliser le terme scalabilité.

En ce sens, le terme *scale-out* sera utilisé pour signifier l’ajout d’unité de manière horizontale à un ensemble, tel que l’ajout de serveur dans un datacenter ou un cluster.

Alors que le terme *scale-up* sera utilisé pour mettre en avant une montée en charge verticale, c’est à dire la mise en place d’une stratégie de croissance en cas de forte demande extérieure.

Ainsi, cette notion de mise à l’échelle désigne la capacité d'un produit à s'adapter à un changement d'ordre de grandeur de la demande (montée en charge), en particulier sa capacité à maintenir ses fonctionnalités et ses performances en cas de forte demande, telle qu’une augmentation significative du nombre d’utilisateurs. Dans ces cas de changement de contexte d’exécution, le système doit continuer de fonctionner de manière nominale, peu importe l’ordre de grandeur du paramètre considéré.

En outre, dans de nombreux cas d’utilisation, une partie de l'application nécessitera toujours plus de ressources que le reste. Par exemple, ce sera le cas pour une application monolithique, pour laquelle il sera donc nécessaire d’augmenter les ressources utilisées pour toute l'application, si une montée en charge survient.

Les microservices ne présentent pas cet inconvénient, puisque chaque service se voit accorder des ressources de manière indépendante. Il est donc tout à fait possible d'augmenter les ressources uniquement pour les services concernés. Les microservices permettent donc d'avoir une gestion plus fine des ressources disponibles et peuvent réduire le coût d'une montée en charge verticale. Aussi, ajouter de nouvelles fonctionnalités à une application basée sur les microservices se fait simplement en ajoutant de nouveaux services.

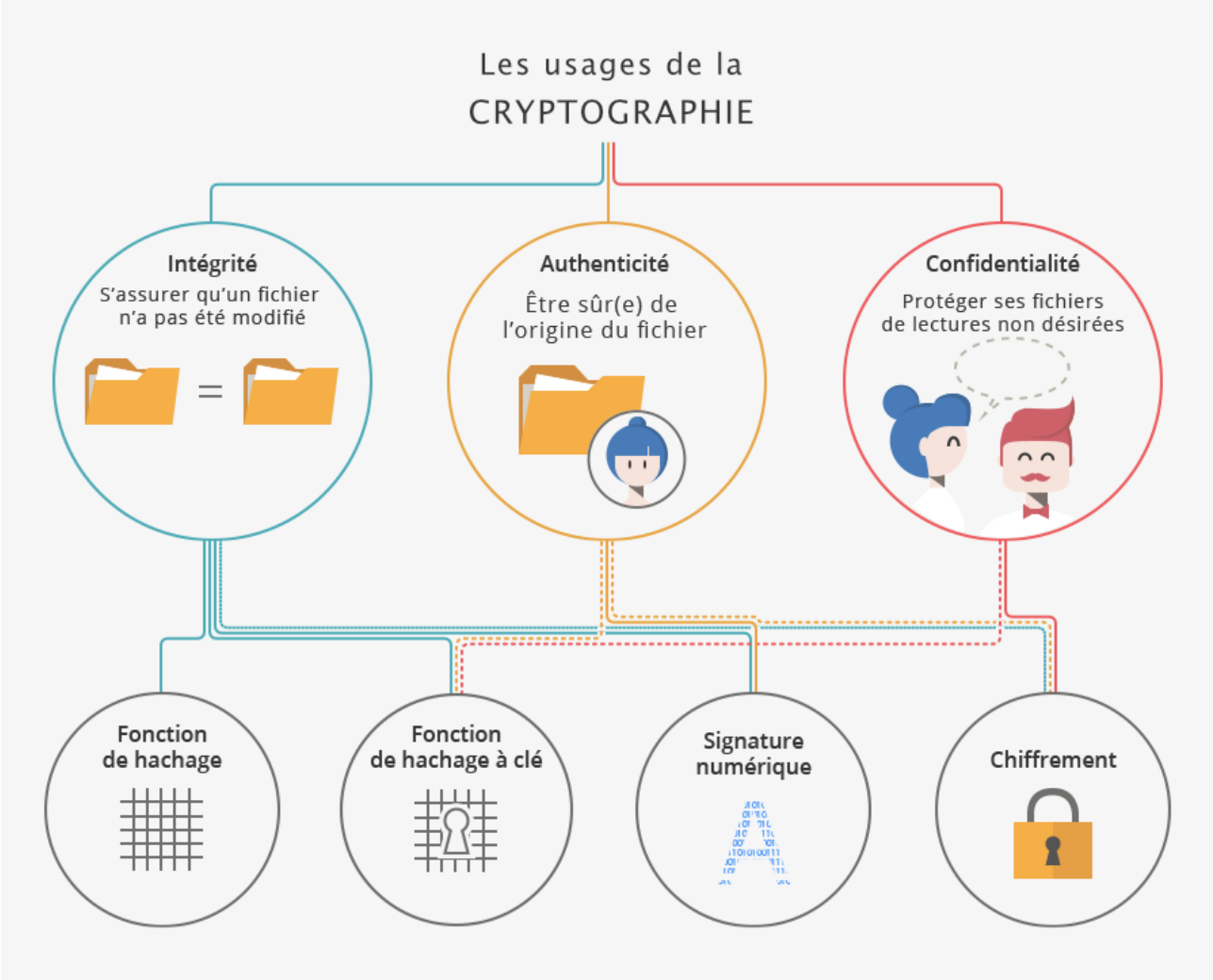
Autrement dit, les microservices peuvent monter en charge (en anglais *scale-out*) de façon indépendante. Au lieu d’avoir une seule application monolithique à monter en charge en totalité, les microservices montent en charge spécifiquement et indépendamment des autres microservices constituant le système. Il est alors possible d’effectuer un *scale-out* ciblé de la zone fonctionnelle qui nécessite plus de puissance de traitement ou plus de bande passante réseau pour satisfaire la demande, au lieu d’effectuer un *scale-out* d’autres zones de l’application qui n’en ont pas besoin.

Cela aura pour conséquence des frais réduits puisque moins de ressources/matériels seront sollicités.

# Sécurité

Ce document se doit de faire un focus sur le module de chiffrement qui a été ajouté dans le *TBA*, pour comprendre la motivation de cet ajout au travers des grands principes de la cryptologie et du chiffrement.

Historiquement, la cryptologie correspond à la science du secret, c'est-à-dire au chiffrement. Aujourd'hui, elle s'est élargie au fait de prouver qui est l'auteur d'un message et s'il a été modifié ou non, grâce aux signatures numériques et aux fonctions de hachage.




Étymologiquement, la cryptologie est la science du secret. Elle réunit la cryptographie (« *écriture secrète* ») et la cryptanalyse (étude des attaques contre les mécanismes de cryptographie).

La cryptologie ne se limite pas à assurer la **confidentialité** des secrets. Elle s'est élargie au fait d'assurer mathématiquement d'autres notions telles que l'**authenticité** d'un message (qui a envoyé ce message ?) ou l'**intégrité** (est-ce que le message a été modifié ?).

Pour assurer ces usages, la cryptologie regroupe quatre principales fonctions :

- le hachage avec clé ;
- le hachage sans clé ;
- la signature numérique ;
- le chiffrement.

# Le hachage

La cryptologie permet de détecter si le message, ou l'information, a été involontairement modifié.

Ainsi, une *fonction de hachage* permettra d'associer à un message, à un fichier ou à un répertoire, une empreinte unique calculable et vérifiable par tous. Cette empreinte est matérialisée par une longue suite de chiffres et de lettres précédées du nom de l'algorithme utilisé, par exemple *SHA2* ou *SHA256*.

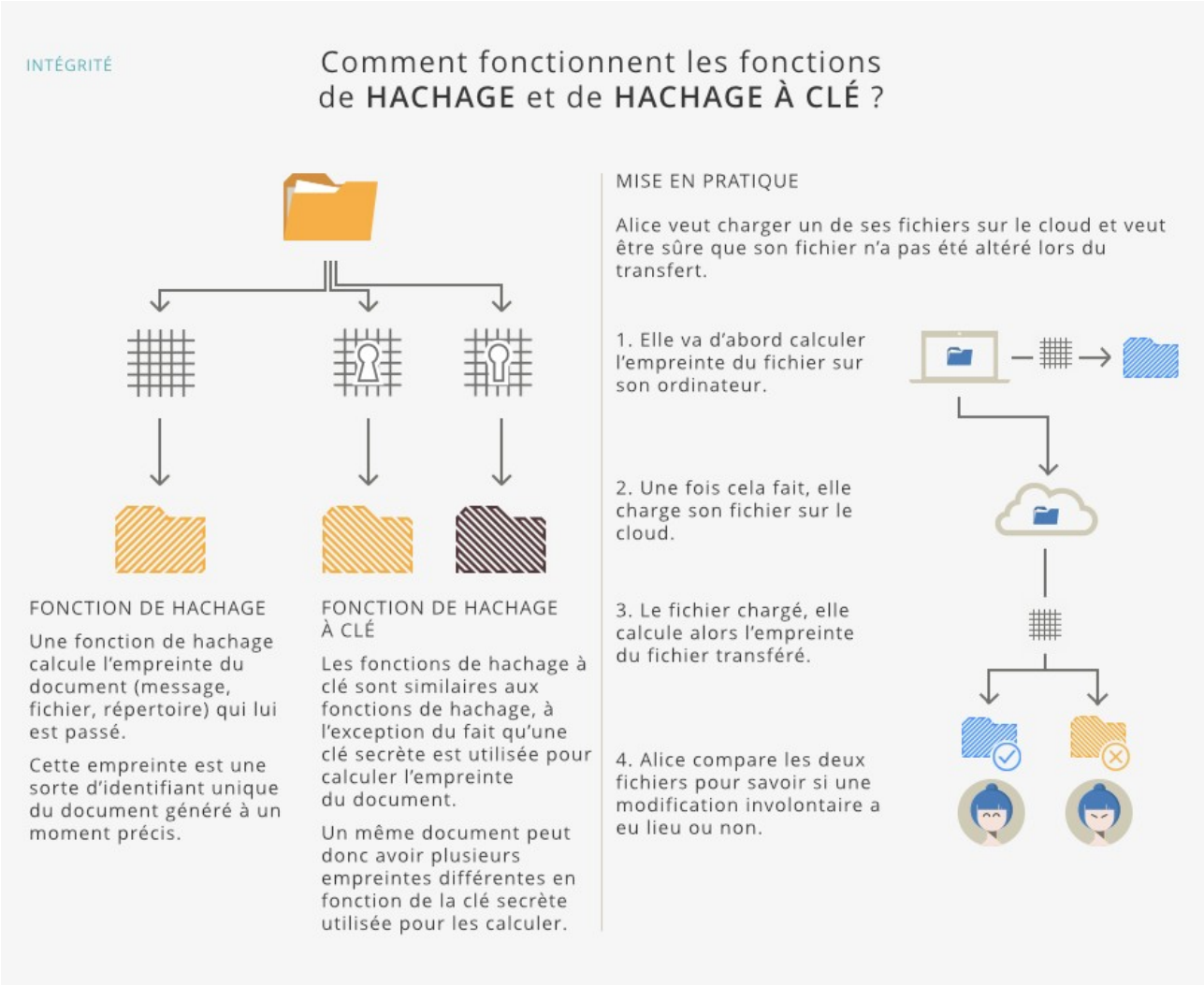
Il ne faut toutefois pas confondre le chiffrement, qui permet d'assurer la confidentialité, c'est-à-dire que seules les personnes autorisées peuvent y avoir accès, et le hachage qui permet de garantir que le message est intègre, c'est-à-dire qu'il n'a pas été modifié.

Ces fonctions de hachage peuvent donc servir, par exemple, à sauvegarder des données sur un espace d'hébergement et vérifier que le téléchargement s'est bien déroulé...

De plus, cette fonctionnalité peut être utiliser, de manière indirecte, pour synchroniser des dossiers, détecter ceux qu'il faut sauvegarder à nouveau et ceux qui n'ont pas été modifiés…

Il existe aussi des « **fonctions de hachage à clé** » qui permettent de rendre le calcul de l'empreinte différent en fonction de la clé utilisée. Avec celles-ci, pour calculer une empreinte, on utilise une clé secrète. Pour deux clés différentes l'empreinte obtenue sur un même message sera différente. Donc pour qu'Alice et Bob calculent la même empreinte, ils doivent tous les deux utiliser la même clé.

C'est parmi ces fonctions de hachage à clé que l'on trouve celles utilisées pour stocker les mots de passe de façon sécurisée.




# La signature

Au même titre que pour un document administratif ou un contrat sur support papier, le mécanisme de la **signature** numérique permet de vérifier qu'un message a bien été envoyé par le détenteur d'une *clé publique*. Ce procédé cryptographique permet à toute personne de s'assurer de l'identité de l'auteur d'un document et permet en plus d'assurer que celui-ci n'a pas été modifié.

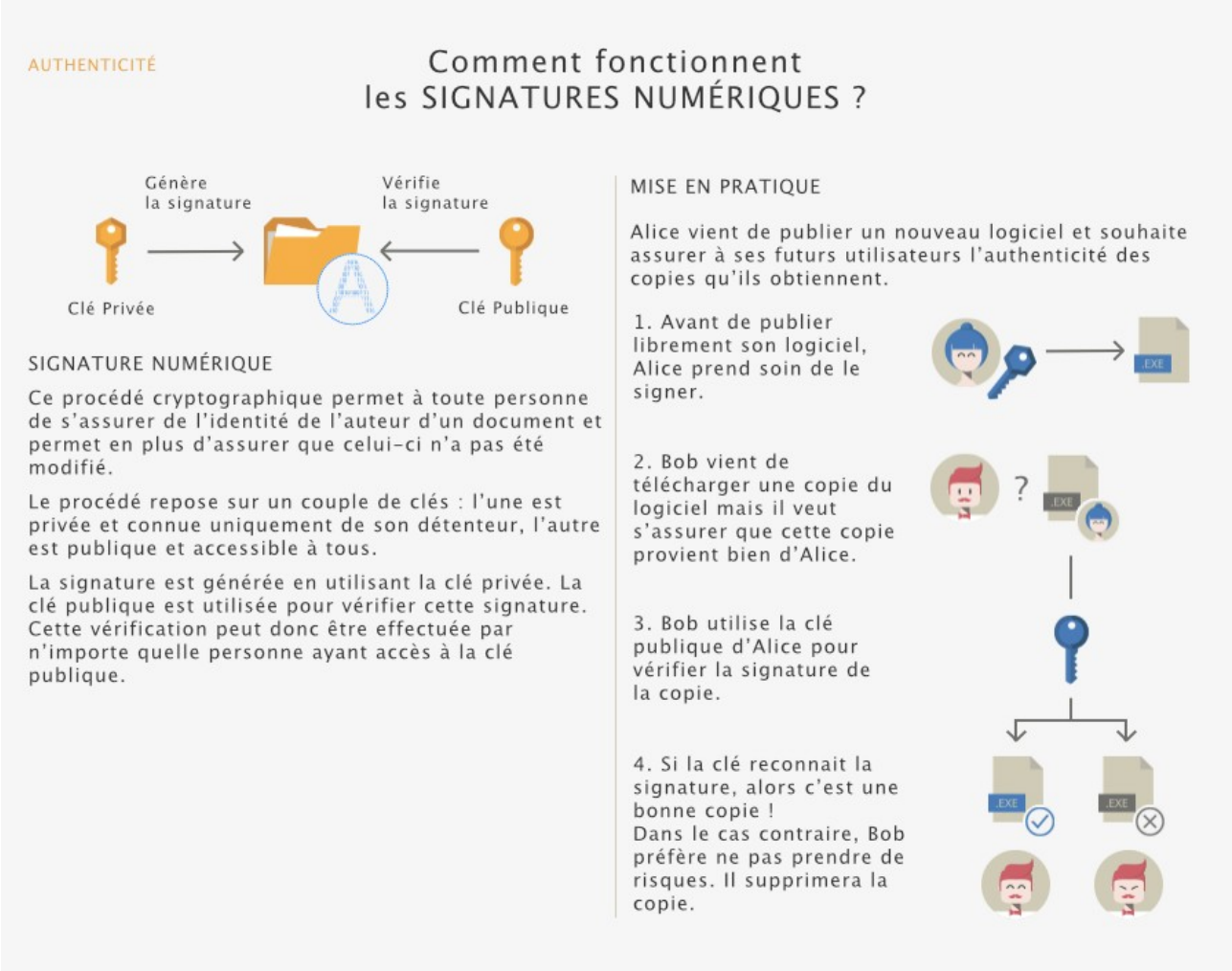
Ainsi, l'utilisation d'une clef publique peut permettre de garantir l'identité d'un émetteur d'un courriel et s'assurer qu'une information provient d'une source sûre.

Concrètement, pour pouvoir signer son document, Alice doit se munir d'une paire de clés :

- l'une, dite *publique*, qui peut être accessible à tous, et en particulier à Bob qui est le destinataire des messages qu'envoie Alice ;
- l'autre, dite *privée*, qui ne doit être connue que d'Alice.

En pratique, Alice génère sa signature avec sa clé privée qui n’est connue que d’elle.

N’importe quelle personne ayant accès à la clé publique d’Alice, dont Bob, peut vérifier la signature sans échanger de secret.




# Le chiffrement

Le chiffrement d’un message permet de garantir que seuls l’émetteur et le(s) destinataire(s) légitime(s) d’un message en connaissent le contenu. C’est une sorte d’enveloppe numérique scellée.

Une fois chiffré, faute d'avoir la clé spécifique, un message est inaccessible et illisible, que ce soit par les humains ou les machines. Cela permet de s’assurer que seul le destinataire ait accès au message et que les informations à envoyer sous enveloppe numérique resteront illisibles par tous.

Il existe deux grandes familles de chiffrement :

- Le **chiffrement symétrique** qui permet de chiffrer et de déchiffrer un contenu avec la même clé, appelée alors la *clé secrète*. Ce type de chiffrement est particulièrement rapide mais nécessite que l'émetteur et le destinataire se mettent d'accord sur une clé secrète commune ou se la transmettent par un autre canal. Celui-ci doit être choisi avec précautions, sans quoi la clé pourrait être récupérée par les mauvaises personnes, ce qui n'assurerait plus la confidentialité du message ;

- Le **chiffrement asymétrique** qui suppose que le (futur) destinataire est muni d'une paire de clés (*clé privée*, *clé publique*) et qu'il a fait en sorte que les émetteurs potentiels aient accès à sa clé publique. Dans ce cas, l'émetteur utilise la clé publique du destinataire pour chiffrer le message tandis que le destinataire utilise sa clé privée pour le déchiffrer.

Parmi ses avantages, la clé publique peut être connue de tous et publiée. Néanmoins, il faut porter une attention toute particulière au fait qu'il est nécessaire que les émetteurs aient confiance en l'origine de la clé publique, et qu'ils soient sûrs qu'il s'agit bien de celle du destinataire.

Autre point fort du chiffrement asymétrique : il n'est plus nécessaire de partager une même clé secrète ! Le chiffrement asymétrique permet de s'en dispenser. Néanmoins, il demeure plus lent que son homologue symétrique.

Pour cette dernière raison, il existe une technique combinant chiffrements « symétrique » et « asymétrique », connue sous le nom de **chiffrement hybride**.

Cette fois, une clé secrète est déterminée par une des deux parties souhaitant communiquer et celle-ci est envoyée chiffrée par un chiffrement asymétrique. Une fois connue des deux parties, celles-ci communiquent en chiffrant symétriquement leurs échanges. Cette technique est notamment appliquée lors de la visite des sites dotés du protocole **https**.


Astra Recherche