

Architecture Microservices dans Azure avec Azure Service Fabric



Par Thomas Rannou

thomasrannou.azurewebsites.net

Table des matières

1. INTRODUCTION	5
1.1. OBJET DU DOCUMENT	5
1.2. ORGANISATION DU DOCUMENT	5
2. PRESENTATION ARCHITECTURE MICROSERVICES	6
2.1. LES ARCHITECTURES DE SERVICES	7
2.1.1. <i>Service Oriented Architecture</i>	7
2.1.2. <i>Web Oriented Architecture</i>	9
2.1.2.1. Spécificités REST	9
2.1.2.2. RESTFUL	10
2.1.3. <i>SOA vs WOA</i>	11
2.2. ARCHITECTURE MICROSERVICES	13
2.2.1. <i>Avantages</i>	14
2.2.2. <i>Inconvénients</i>	15
2.2.3. <i>Prérequis</i>	16
3. REFLEXION AUTOUR D'UNE MISE EN ŒUVRE CONCRETE	17
3.1. TECHNOLOGIES	17
3.1.1. <i>WCF SOAP ou Web API REST</i>	17
3.1.2. <i>.Net Core ou .Net Framework</i>	18
3.2. TECHNIQUES	19
3.2.1. <i>Définition du périmètre d'un Microservices</i>	19
3.2.2. <i>Problématiques à étudier</i>	19
3.2.2.1. Communication clients / services	19
3.2.2.2. La communication entre services	21
3.2.2.3. Gestion des données	22
3.2.2.4. Gestion des logs et monitoring	25
3.2.2.5. Gestion des pannes et résilience	25
3.2.2.6. Gestion des transactions	26
3.2.3. <i>Application Lifecycle Management</i>	26
3.2.3.1. Intégration	27
3.2.3.2. Tests	28
3.2.3.3. Déploiements	30
3.3. HEBERGEMENT DANS AZURE	31
3.3.1. <i>Présentation générale</i>	31
3.3.2. <i>Web App</i>	32
3.3.3. <i>Azure Container Service (ACS)</i>	33
3.3.4. <i>Azure Service Fabric</i>	34
3.3.4.1. Communication clients / services	37
3.3.4.2. Communication entre services	39
3.3.4.3. Gestion des données	44
3.3.4.4. Gestion des logs et monitoring	44
3.3.4.5. Gestion des pannes et résilience	48
3.3.4.6. Gestion des transactions	49
4. MISE EN ŒUVRE	50
4.1. EXISTANT	50
4.2. PROBLEMATIQUES DE L'EXISTANT	50
4.3. CONCEPTIONS	51
4.4. IMPLEMENTATIONS	56
4.4.1. <i>Architecture d'un service</i>	56
4.4.2. <i>Création et configuration de Service Fabric</i>	57
4.4.3. <i>Projet Visual Studio Service Fabric</i>	67
4.4.3.1. Création du projet	67

4.4.3.2.	Tests en local	70
4.4.3.3.	Premiers déploiements	71
4.4.3.4.	Communication avec Service Fabric.....	72
4.5.	APPLICATION LIFECYCLE MANAGEMENT	75
4.5.1.	<i>Intégrations</i>	76
4.5.2.	<i>Tests</i>	78
4.5.3.	<i>Déploiements</i>	78
4.6.	ANALYSE.....	80
5.	CONCLUSION	82
ANNEXE A.	ARTICLES ET TUTORIELS UTILISES	83
ANNEXE B.	VIDEOS.....	89
ANNEXE C.	LIVRES	90

1. Introduction

1.1. Objet du document

Le présent document constitue une étude sur les architectures de type microservices et leur possible mise en œuvre dans la plateforme Azure.

1.2. Organisation du document

Le rapport est divisé en quatre chapitres :

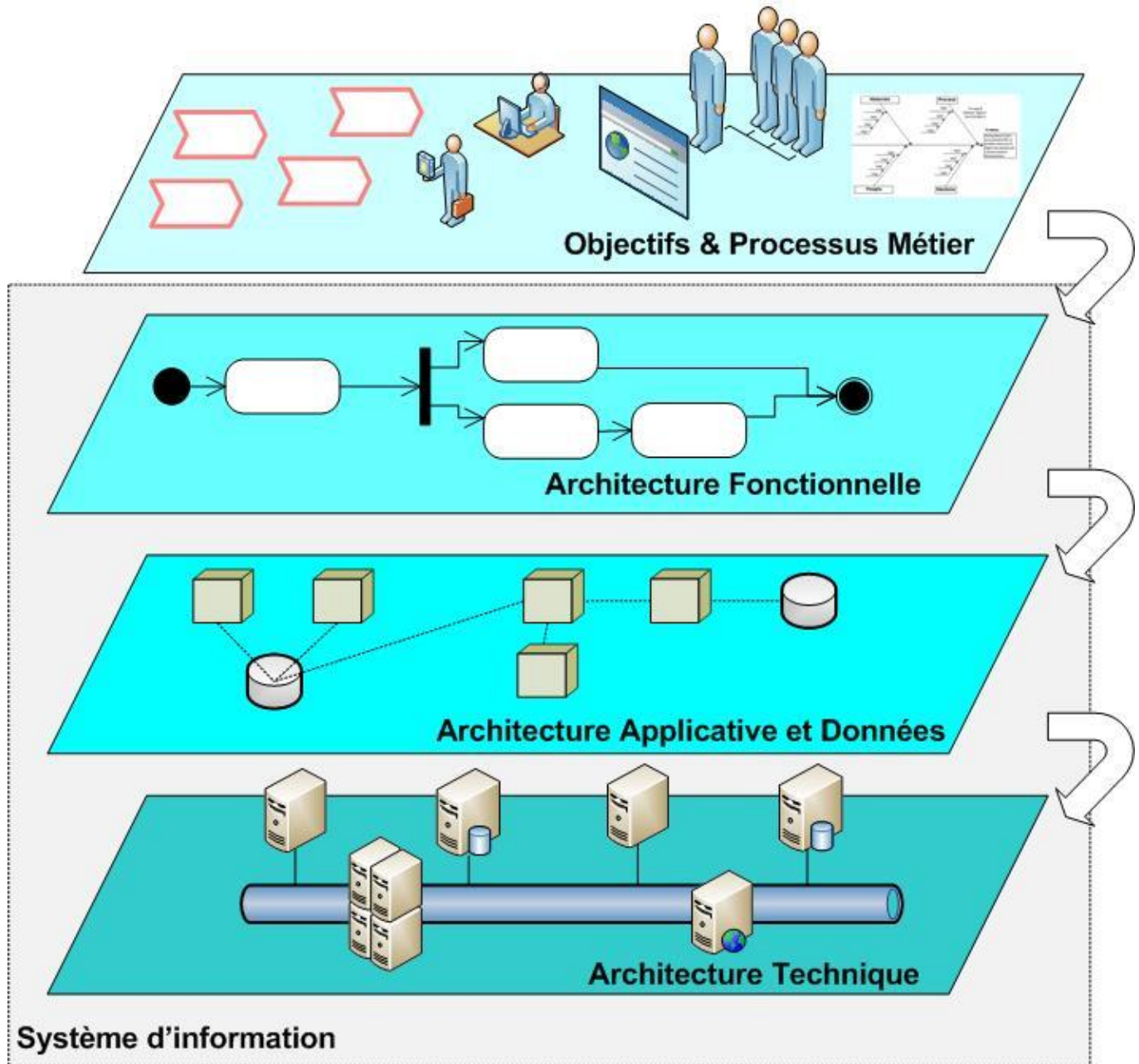
- Le chapitre 1 (présent chapitre) constitue l'introduction du document ;
- Le chapitre 2 présente les différentes architectures de services existentes ;
- Le chapitre 3 présente une réflexion autour de l'implémentation de microservices ;
- Le chapitre 4 expose une implémentation concrète.

Les annexes fournissent des informations complétant le rapport.

Les liens utilisés pour la constitution de ce document sont également en annexe.

2. Présentation architecture Microservices

Ce chapitre a pour objet les architectures applicatives présentes dans les Systèmes d'Information. Des systèmes SOA classiques nous verrons comment l'essor du Cloud bouleverse les écosystèmes applicatifs on les rendant plus autonome, évolutif et indépendant des autres composants du système.



2.1. Les architectures de services

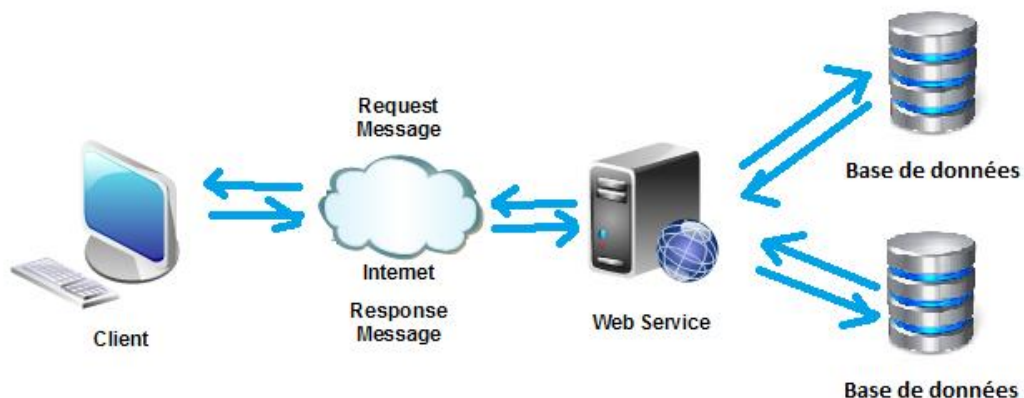
2.1.1. Service Oriented Architecture

Un Service est un composant logiciel distribué, exposant les fonctionnalités d'un domaine métier. Cette définition, la plus synthétique possible d'un service, peut être accompagnée de ces 8 caractéristiques proposées par Thomas Erl dans son livre « SOA Principles of Service Design » :

- Contrat standardisé
- Couplage lâche
- Abstraction
- Réutilisabilité
- Autonomie
- Sans état
- Découvrabilité
- Composabilité

Le but de ce style d'architecture apparu dans les années 2000 est de fournir au SI un cadre d'écriture de composant logiciel :

- Réutilisable
- Evolutif
- Urbanisé autour de domaines fonctionnels
- Modulaire



Pour mettre en œuvre cette architecture SOA, les entreprises ont rapidement adoptés le format Webservices au travers du protocole SOAP et du langage de description WSDL. Celui-ci permet la description des services offerts par le Web Service et SOAP définit le protocole d'échange entre un client et un fournisseur de service, le plus souvent au-dessus du protocole HTTP.

Présentation de SOAP

SOAP (Simple Object Access Protocol) est un protocole défini à l'origine par Microsoft, puis standardisé par le W3C, utilisant la notation XML permettant de définir les mécanismes d'échanges d'information entre des clients et des fournisseurs de services web.

Présentation de WSDL

Le standard WSDL (Web Service Description Language) est un langage reposant sur la notation XML permettant de décrire les services web. WSDL permet ainsi de décrire l'emplacement du service web ainsi que les opérations (méthodes, paramètres et valeurs de retour) que le service propose.

Les principaux avantages de ce protocole sont :

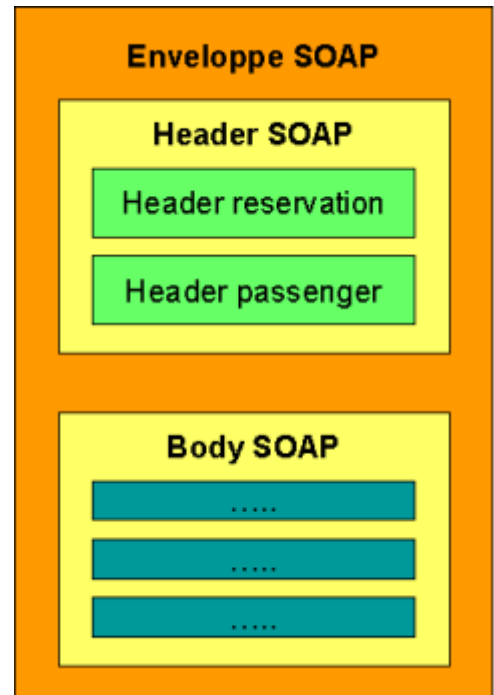
- L'indépendance vis à vis du langage de programmation
- L'indépendance vis à vis de la plateforme où ils sont déployés
- Son extensibilité
- L'utilisation du standard XML pour l'échange des messages
- Le contenu des messages échangés qui incluent le format d'échange cela permet d'avoir un contrat personnalisé entre le fournisseur et le consommateur du service
- SOAP est indépendant de la couche de transport puisqu'il l'a redéfini. HTTP est le protocole de transport le plus largement utilisé par SOAP.

Cette façon de mettre en oeuvre une SOA s'est souvent faite en utilisant un composant central de médiation entre clients et services : l'ESB, Entreprise Service Bus. Le rôle de ce composant est de :

- Découpler consommateurs et fournisseurs de services : abstraction des protocoles de communications, des langages.
- Tracer les échanges
- Agréger les services
- Mutualiser les accès : gestion des droits, transformation des données.

Cette façon d'implémenter une SOA est majoritaire en entreprise mais la complexité de la mise en oeuvre de SOAP et la verbosité du protocole, entre autre, ont favorisé l'apparition d'une autre façon d'implémenter une SOA, basée sur les travaux de Roy Fielding.

Sa thèse « Architectural Styles and the Design of Network-based Software Architectures. » publié en 2000 et notamment son travail sur REST ont favorisé l'émergence d'une nouvelle manière de mettre en oeuvre une architecture de service.



2.1.2. Web Oriented Architecture

Les architectures des Géants du Web, presque exclusivement basées sur le style REST, ont été longtemps opposées aux architectures SOA d'entreprises, majoritairement basées sur SOAP. Mais les APIs REST sont une forme de SOA, dont l'objectif est d'utiliser HTTP comme protocole applicatif. On parle alors d'architectures orientées Web : WOA.

L'objectif de cette architecture est de proposer une version plus simple de la SOA, utilisant les caractéristiques et les standards du Web, plutôt que de chercher à les abstraire. L'architecture REST utilise les spécifications originelles du protocole HTTP, plutôt que de réinventer une surcouche comme SOAP :

- L'URI comme identifiant des ressources.
- Les verbes HTTP comme identifiant des opérations.
- Les réponses HTTP comme représentation des ressources (HTML, XML, CSV, JSON, ...).

REST est donc fortement recommandé pour des cas simples où on cherche à effectuer des actions sur un contenu, comme lister des ressources, accéder à une ressource, en créer de nouvelles, modifier une ressource existante, ou en supprimer une. Pour toutes ces actions on utilisera la commande HTTP adéquat :

- Lire une ressource, ou une collection de ressources (GET)
- Modifier une ressource existante (PUT)
- Créer une ressource (POST)
- Supprimer une ressource (DELETE)

Exemple :

- Un GET `http://www.demo.com/clients` signifie que je souhaite récupérer la liste des clients disponibles.
- Un GET `http://www.demo.com/clients/2` signifie que je souhaite récupérer les informations du client dont l'identifiant est 2.
- Au même titre, un DELETE `http://www.demo.com/clients /2` devrait supprimer le client 2.

2.1.2.1. Spécificités REST

Les Propriétés d'une API REST sont :

Uniformité : Chaque ressource est identifiée de façon unique par son URL. L'interface est uniforme à tous les niveaux. Tous les éléments communiquent en utilisant la même interface.

Sans état : Une API REST ne doit pas maintenir de session. Cela évite entre autres, les problèmes de loadbalancing par exemple et permet de garantir des appels idempotents.

Mise en cache : Il doit être possible d'utiliser les implémentations standards de cache HTTP.

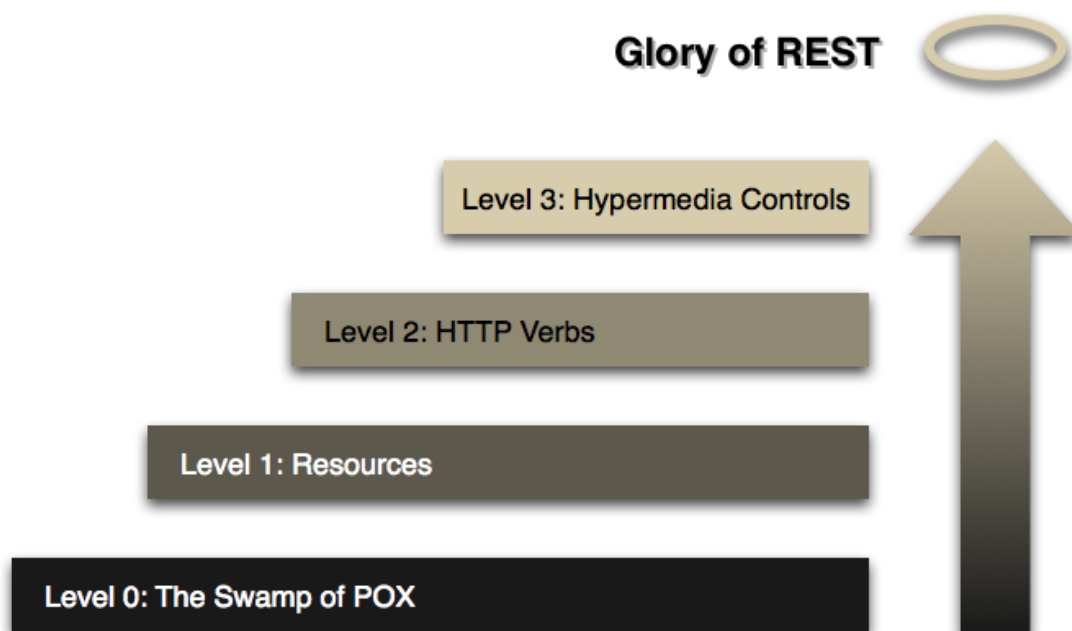
Client / Serveur : Separation of Concerns : L'API ReST n'est pas concernée par l'affichage, les interactions utilisateur et la session.

Layered : La présence de "connecteurs" intermédiaires doit être implicite pour le client et le serveur (composant de cache / sécurité etc...).

Pour obtenir des conseils sur la mise en œuvre d'une API REST, consultez : <https://blog.octo.com/designer-une-api-rest/> qui résume plutôt bien les différentes problématiques à étudier avant de se lancer dans l'implémentation.

2.1.2.2. RESTFUL

En se basant sur le travail de Roy Fielding, Leonard Richardson a établi un modèle de maturité des services web REST appelé Richardson Maturity Model (RMM). Ce modèle est composé de quatre niveaux permettant d'évaluer une API par rapport aux contraintes REST.



- **Niveau 0** : Le protocole HTTP est utilisé uniquement à des fins de transport du message. L'ensemble des données transitent par un seul et unique point d'entrée.
- **Niveau 1** : Introduction de la notion de ressources. Il y a donc désormais plusieurs endpoint (URI) par ressource.
- **Niveau 2** : Apparition de la notion d'action et d'état, ce qui correspond aux verbes, GET, POST, PUT, DELETE et aux codes http (200, 404, 500...).

- **Niveau 3** : Ultime et dernière notion de REST : HATEOAS (Hypertext As The Engine Of Application State). Notre API est « autodocumentée », c'est-à-dire que l'on peut passer d'une action à une autre via des URL transmises par l'API dans les réponses.

Une API RESTful est une API qui respecte toutes les contraintes REST. La très grande majorité du temps, les API sont au niveau 2 du RMM.

2.1.3. SOA vs WOA

Les deux approches, SOA et WOA, sont donc envisageables pour la mise en œuvre d'une architecture de services, et présentent toute deux des points forts mais également des points faibles et possèdent en fin de compte une utilisabilité propre à des environnements différents. Nous les détaillerons un peu plus loin.

Tout d'abord, pour bien comprendre les avantages et inconvénients présentés plus bas voici un exemple d'appel à une méthode d'addition de deux nombres :

REST

La requête

```
GET https://demo.com/sum?a=40&b=2
```

La réponse

42

La documentation

La ressource « somme » est la somme de deux paramètres entiers *a* et *b*, représentée par une chaîne de caractères.

SOAP

La requête

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<soapenv:Body>
<ns0:additionner
xmlns:ns0="http://demo.com"
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<valeur1 xsi:type="xsd:int">40</valeur1>
<valeur2 xsi:type="xsd:int">2</valeur2>
</ns0:additionner>
</soapenv:Body>
</soapenv:Envelope>
```

La réponse

```

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<soapenv:Body>
<ns1:additionerResponse
xmlns:ns1="http://demo.com"
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<additionerReturn href="#id0" />
</ns1:additionerResponse>
<multiRef xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
id="id0" soapenc:root="0"
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xsi:type="xsd:long">42</multiRef>
</soapenv:Body>
</soapenv:Envelope>

```

La documentation (WSDL)

```

<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definition targetNamespace="http://axis.test.com"
xmlns:apacheSOAP="http://xml.apache.org/xml-soap"
xmlns:impl="http://axis.test.com"
xmlns:intf="http://axis.test.com"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<!--WSDL created by Apache Axis version: 1.3
Built on Oct 05, 2005 (05:23:37 EDT)-->
<wsdl:message name="additionerRequest">
<wsdl:part name="valeur1" type="xsd:int"/>
<wsdl:part name="valeur2" type="xsd:int"/>
</wsdl:message>
<wsdl:message name="additionerResponse">
<wsdl:part name="additionerReturn" type="xsd:long"/>
</wsdl:message>
<wsdl:portType name="Calculer">
<wsdl:operation name="additioner" parameterOrder="valeur1 valeur2">
<wsdl:input message="impl:additionerRequest" name="additionerRequest"/>
<wsdl:output message="impl:additionerResponse" name="additionerResponse"/>
</wsdl:operation>
</wsdl:portType>
<wsdl:binding name="CalculerSoapBinding" type="impl:Calculer">
<wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
<wsdl:operation name="additioner">
<wsdlsoap:operation soapAction="" />
<wsdl:input name="additionerRequest">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://demo.com" use="encoded"/>
</wsdl:input>
<wsdl:output name="additionerResponse">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://demo.com" use="encoded"/>
</wsdl:output>
</wsdl:operation>
</wsdl:binding>
<wsdl:serviceName="CalculerService">
<wsdl:port binding="impl:CalculerSoapBinding" name="Calculer">
<wsdlsoap:address location="http://demo.com">
</wsdl:port>

```

```
</wsdl:service>  
</wsdl:definitions>
```

Comparons maintenant les deux approches.

Les avantages de REST

- Simplicité d'utilisation et de développement
- Opérations CRUD « native »
- Consommable par tous supports puisque basé sur HTTP.
- Plus souple

Les avantages de SOAP

- Sécurité et fiabilité du protocole
- Contrats formels entre le client et le serveur
- Opérations avec état : partage de transaction possible
- Possibilité de communiquer avec un autre protocole que HTTP (TCP/IP)

De façon synthétique, REST propose une mise en œuvre simple de développement et d'utilisation, basé sur des standards connus de tous. SOAP en revanche sera utile si les contraintes de sécurité sont importantes et que le contrat entre client et services doit être absolu.

Nous avons présenté deux façons d'implémenter une architecture de services dans une entreprise, la première basée sur le protocole de communication SOAP est de moins en moins utilisé au détriment de la seconde basé sur REST.

Une dernière implémentation est apparue récemment, adapté au contrainte d'agilité des SI et l'émergence du Cloud.

2.2. Architecture Microservices

« In short, the microservice architectural style is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API. These services are built around business capabilities and independently deployable by fully automated deployment machinery. There is a bare minimum of centralized management of these services, which may be written in different programming languages and use different data storage technologies. »

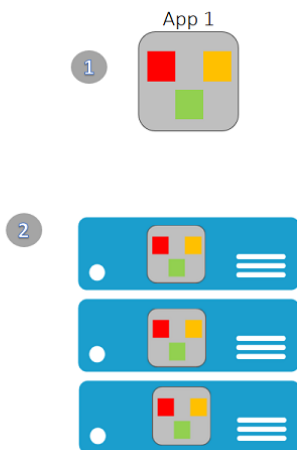
MARTIN FOWLER

Les architectures Microservices introduit par Martin Fowler dans son article : <https://martinfowler.com/articles/microservices.html> vont nous permettre dans un premier temps de relever les grandes idées derrière le terme de « microservice » :

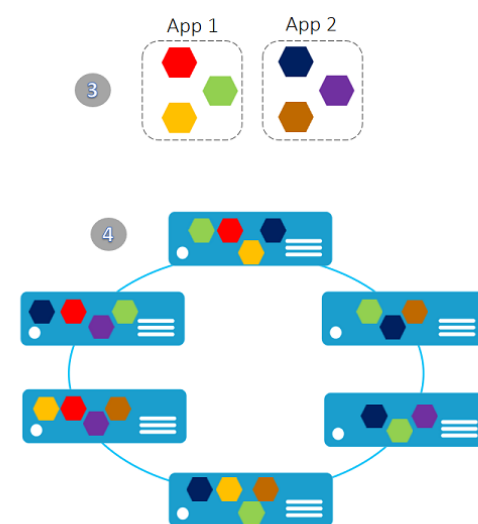
- Un composant logiciel correspondant à un besoin fonctionnel précis, définis et délimité.
- Un composant autonome au niveau du déploiement et de l'exécution.

Un microservice est donc une unité de service fonctionnelle qui se développe, se déploie, s'exécute et gère ses données indépendamment des autres services du système.

Monolithic application approach



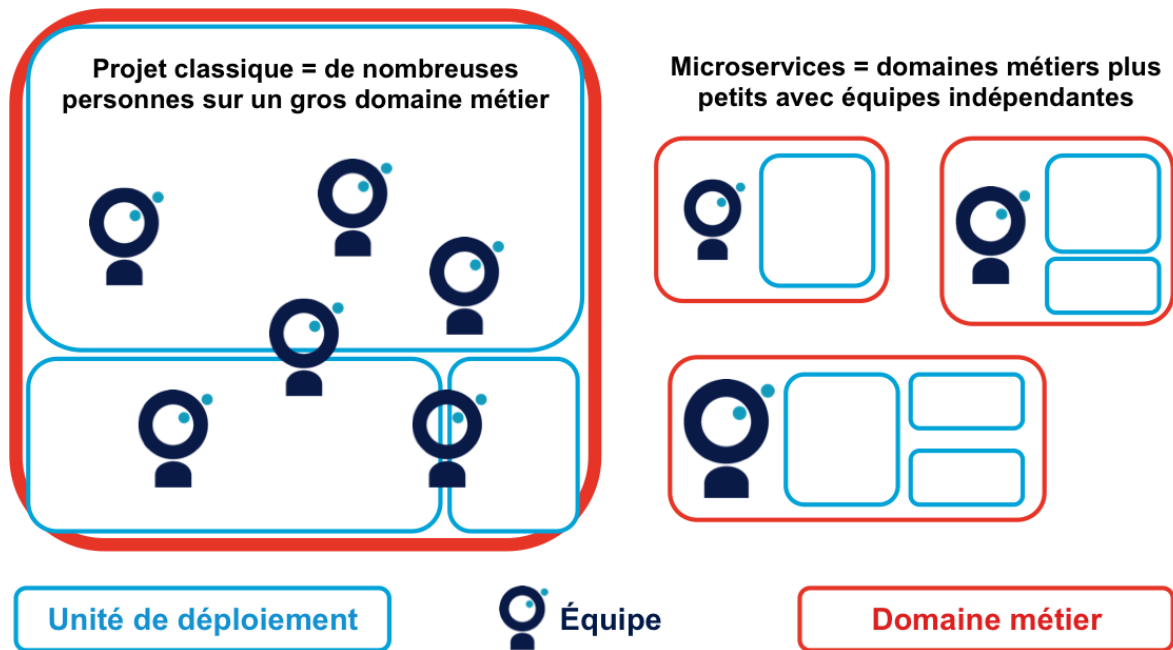
Microservices application approach



Avec une application monolithique, si on veut déployer une application sur un nouveau serveur il faut tout dupliquer. Avec une application en microservices, vous pouvez déployer chaque service sur 1 ou n serveurs, indépendamment des autres.

2.2.1. Avantages

- Code base indépendant : Les services sont plus simples à appréhender à développer et à repenser.
- Les services peuvent être développés et déployés de façon plus indépendante et plus rapide.
- Les services démarrent et s'arrêtent rapidement, ce qui réduit le temps de déploiement et améliore la productivité des développeurs.
- Il est plus simple de mettre uniquement les composants nécessaires à l'échelle de façon dynamique pour répondre à la demande.
- Le système tolère mieux les pannes pour une disponibilité plus élevée.
- Les microservices peuvent être mis à niveau individuellement, en temps réel et sans interrompre le service.
- Indépendances et agilité technologiques entre microservices.
- Séparation des responsabilités.



La contrepartie logique de cette architecture c'est le développement, le déploiement et l'administration d'une grande quantité de services. Ainsi, si chaque service est plus simple, le système dans son ensemble est plus complexe.

"Loosely coupled service oriented architecture with bounded contexts"

ADRIAN COCKROFT

2.2.2. Inconvénients

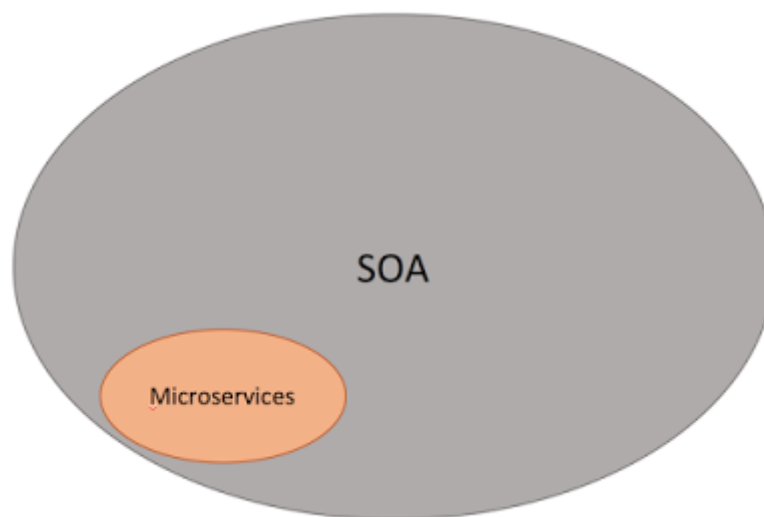
- Augmentation de la complexité du SI.
- Intégration et déploiements plus nombreux.
- Tests plus difficiles si il y a des appels entre microservices.
- Hausse nombre d'éléments à développer, surveiller, livrer.
- Augmentation Traffic réseau et latence des appels HTTP.
- Les microservices peuvent être très hétérogènes ce qui peut rendre leur implémentation plus complexe qu'un monolithe.

Si dans la théorie un microservice est un composant logiciel plutôt simple, sa mise en œuvre suscite bon nombre de questions que nous détaillerons dans notre partie « Mise en œuvre concrète ». Dans tous les cas un certains nombres de besoins sont déjà identifiable.

2.2.3. Prérequis

- Provisionnement de serveurs rapide
- Solution de monitoring de services
- Gestion des logs
- Solution de déploiements et rollback(Blue green déploiement, Canary Release)
- Maturité DevOps
- Gestion de Configuration
- Tests
- Découverte de service pour le loadbalancing

Si nous prenons un peu de recul, on se rend compte qu'au-delà de certaines idées novatrices, la scalabilité et l'isolation des composants notamment, bon nombre des caractéristiques d'une architecture microservices se retrouvent dans une architecture de services plus classique.



Ainsi beaucoup n'hésite pas à qualifier les microservices de « buzzword » alors qu'en fin de compte on parle toujours de la même chose, une architecture de services qui s'adapte aux évolutions de l'IT, au niveau du développement, de l'infrastructure et même du management :

- La SOA a mis en avant les avantages de l'approche services.
- L'agile et le lean startup ont fourni les modèles d'organisation des équipes.
- L'industrialisation des déploiements diminue les coûts de mise en production et d'exploitation.
- L'essor du Cloud qui déporte les infrastructures hors du SI modifie les modèles DevOps.

3. Réflexion autour d'une mise en œuvre concrète

3.1. Technologies

3.1.1. WCF SOAP ou Web API REST

WCF est un Framework de communication de Microsoft. Une application WCF permet de bâtir et de faire communiquer des services basés sur le protocole SOAP. Il succède au service de type ASMX.

Les services WCF permettent de mettre en œuvre une architecture REST mais Microsoft propose depuis 2012 un nouveau type d'application : les Web API.

ASP.Net WebAPI est apparu avec ASP.Net MVC 4. C'est la solution proposée par Microsoft pour permettre aux développeurs de construire rapidement des services web REST. La principale différence entre WebAPI et MVC est que l'un possède des vues alors que l'autre retourne des données sérialisées (JSON/XML).

Avec l'avènement de .NET Core, Microsoft a fusionné les produits ASP.Net MVC et ASP.Net WebAPI. Les classes à manipuler pour développer des projets de type WEB API ou MVC sont maintenant les mêmes !

Microsoft recommande maintenant de bâtir des projets basés sur le modèle ASP .Net Core :

- WebAPI
- MVC
- Angular
- React

MSDN nous indique : « Utilisez WCF pour créer des services Web dignes de confiance et sécurisés accessibles via un large éventail de transports. Utilisez l'API Web ASP.NET pour créer des services HTTP qui sont accessibles à partir d'une large gamme de clients.

Utilisez l'API Web ASP.NET si vous créez et concevez de nouveaux services REST. Bien que WCF prenne en charge l'écriture de services REST, la prise en charge de REST dans l'API Web ASP.NET est plus complète et toutes les futures améliorations des fonctionnalités REST seront apportées dans l'API Web ASP.NET. Si vous disposez d'un service WCF et souhaitez exposer des points de terminaison REST supplémentaires, utilisez WCF et WebHttpBinding. »

Bien sûr au sein d'un SI il est tout à fait possible d'utiliser les deux approches suivant les besoins propres au projet ! WCF est historiquement lié au service SOAP et bien qu'apte à l'implémentation de service REST il me paraît trop lourd pour la réalisation de microservices.

Ici nous utiliserons des services de type Web API Asp .Net Core.

3.1.2. .Net Core ou .Net Framework

Sous l'impulsion de Satya Nadella, autant que sous celle des nécessités d'un cloud Azure ouvert à tous les OS et à tous les langages, le projet « .NET 5 » s'est transformé en un runtime cross-plateforme et open-source connu sous le nom de « .NET Core ».

Pour des soucis de performances .Net Core semble mieux indiqué de par sa légèreté et facilité de déploiement.

Microsoft nous propose :

« *Utilisez .NET Core pour votre application serveur quand :*

- *Vous avez des besoins multiplateformes ;*
- *Vous ciblez des microservices ;*
- *Vous utilisez des conteneurs Docker ;*
- *Vous avez besoin de systèmes scalables et hautes performances.*
- *Vous avez besoin de versions .NET côte à côte par application.*

Utilisez .NET Framework pour votre application serveur quand :

- *Votre application utilise le .NET Framework (nous vous recommandons de privilégier l'extension à la migration).*
- *Votre application utilise des packages NuGet ou des bibliothèques .NET tiers non disponibles pour .NET Core.*
- *Votre application utilise des technologies .NET non disponibles pour .NET Core.*
- *Votre application utilise une plateforme qui ne prend pas en charge .NET Core. »*

Parenthèse sur le projet .Net Standard : Il s'agit d'une spécification pour garantir qu'une librairie de code .Net fonctionne sur toutes les plateformes .Net Standard est donc un jeu d'API que les implémentations .Net (.Net Framework / .Net Core / Xamarin) doivent respecter pour garantir la portabilité de la librairie développée. Ce projet s'inscrit bien sûr dans la volonté de Microsoft de s'ouvrir aux autres plateformes : Linux, MacOS, Android et iOS.

Pour pouvoir développer en .Net Core sur les plateformes autre que Windows, Microsoft nous propose un nouvel IDE, léger, modulaire et OpenSource : <https://code.visualstudio.com/>. En contrepartie, ces fonctionnalités sont bien plus limitées que Visual Studio 2017.

3.2. Techniques

3.2.1. Définition du périmètre d'un Microservices

Un microservice est avant tout une unité fonctionnelle. Il correspond à une fonctionnalité précise, logique et cohérente du système. Un microservice est donc autonome vis à vis de la fonctionnalité qu'il réalise. Il a son propre code, gère ses propres données et ne les partage pas, pas directement en tout cas, avec d'autres services. L'approche nécessite un effort de conception pour faire émerger des unités fonctionnelles autonomes.

Pour parvenir à un résultat cohérent au niveau du service en lui-même et au niveau global du SI (vision de l'urbaniste) plusieurs idées :

- Séparer la complexité fonctionnelle et technique : le service répond à un besoin fonctionnel, pas applicatif !
- Eviter le couplage entre services. S'il y en a trop le contexte est peut être mal délimité !
- Trouver un équilibre entre macroservices et nanoservices

Pour arriver à ce résultat, nous pouvons utiliser l'approche Domain Driven Design d'Eric Evans qui a largement fait ces preuves. DDD, proposé par Evans dès 2003, est une manière de penser la conception autour du code, de collaborer et de communiquer avec les experts fonctionnels et d'avoir une implémentation centrée sur le métier. Le besoin de cohérence fonctionnelle très forte au niveau du découpage de microservices se recoupe très bien avec l'approche DDD.

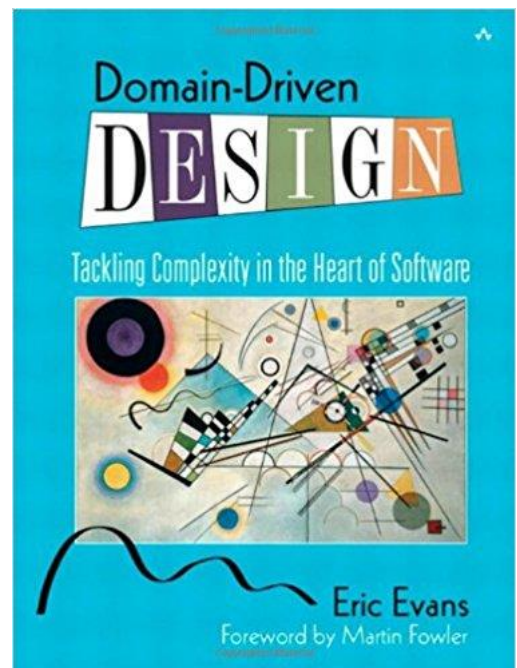
Pour une introduction complète sur le sujet, consulter : <https://cdiese.fr/domain-driven-design-en-5-min/>

3.2.2. Problématiques à étudier

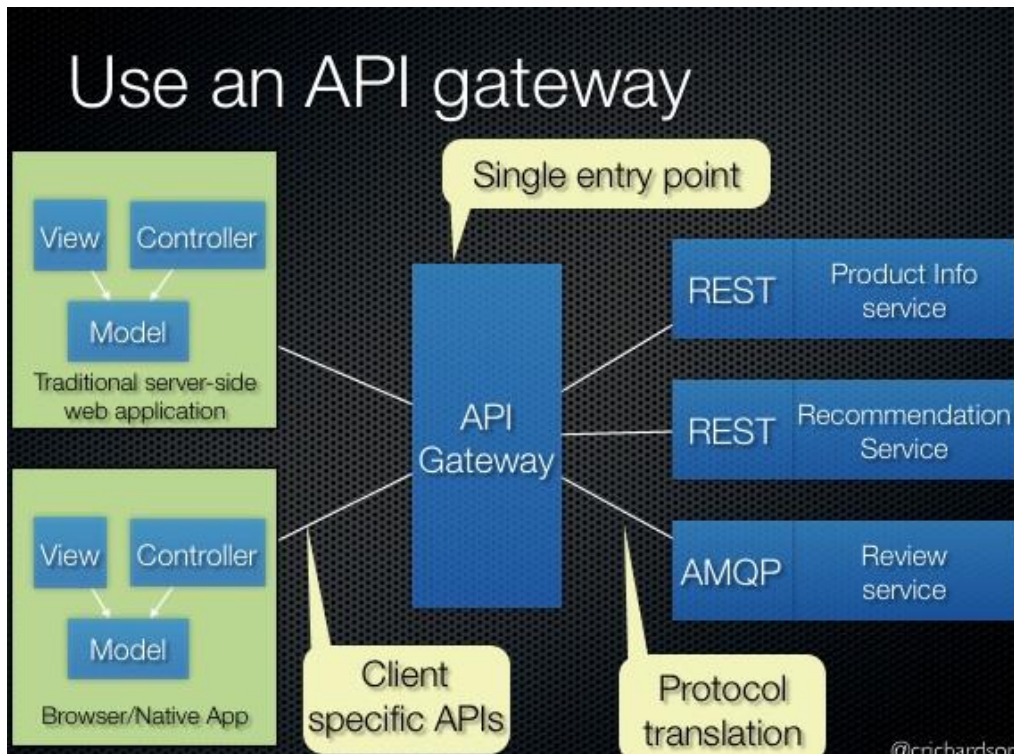
Pour réussir l'implémentation ou la migration vers une architecture microservices, un certain nombre de problématiques techniques sont à étudier en amont, en plus du découpage fonctionnel. Nous présenterons ici les éléments à étudier et les différentes possibilités pour y répondre.

3.2.2.1. Communication clients / services

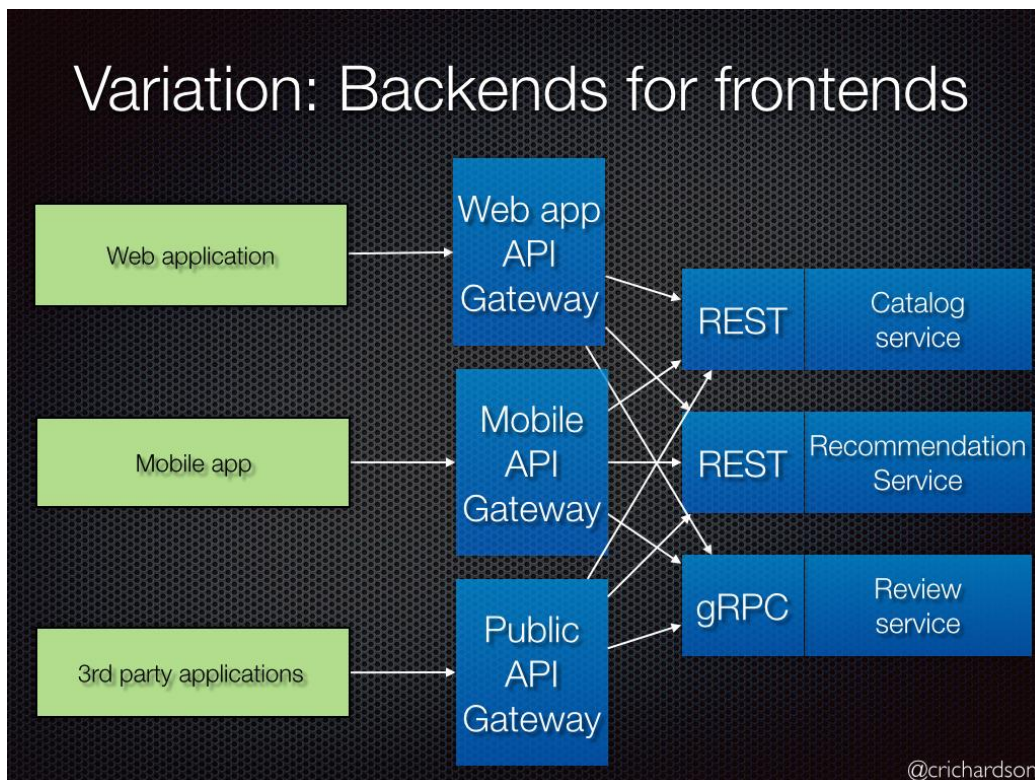
Le consommateur de mon service doit-il avoir un accès direct à celui-ci ou faut-il utiliser un composant de médiation intermédiaire comme un API Gateway ?



Un API Gateway est un composant logiciel qui expose nos services à un client. Ainsi le client n'a d'échange qu'avec la passerelle et pas directement avec nos services.



L'API Gateway peut même être découpé pour éventuellement adresser les clients différents. On parle alors de modélisation « Backends for frontends ».



L'utilisation de la passerelle d'API offre les avantages suivants :

- Il dissocie les clients des services. Les services peuvent être versionnés ou refactorisés sans avoir à mettre à jour tous les clients.
- Les services peuvent utiliser des protocoles de messagerie qui ne sont pas compatibles avec le web, comme AMQP.
- La passerelle d'API peut exécuter d'autres fonctions transverses, telles que l'authentification, la journalisation, la terminaison SSL et l'équilibrage de charge.
- La passerelle peut se charger de la découverte et indexation des services

Attention, la passerelle ne contient pas de fonctionnel, il ne s'agit que d'un « passe plat ».

3.2.2.2. La communication entre services

Les microservices vont forcément communiquer entre eux. Comment doivent-ils procéder ?

Question primordiale à mon sens, en effet la complexité des microservices n'est pas dans leurs code source, mais dans leurs interactions.

Il y a deux possibilités pour faire échanger nos services :

- Communication synchrone
- Communication asynchrone

Aucune n'est meilleure que l'autre et le bon procédé est celui qui correspond à vos besoins.

Effectuer des appels synchrones

Le service client envoie une requête à un second service, il attend pendant le traitement de la requête et il récupère la réponse du service en fin de traitement.

Les appels synchrones sont les plus simples à mettre en œuvre. Les services communiqueront par appels REST et s'échangeront les données dans le format de leur choix (XML / JSON / binaire).

Mais attention, nos microservices doivent être instanciable automatiquement pour répondre à une hausse de la charge par exemple. Compte tenu de leur « volatilité », comment se trouve-t-il sur le réseau s'ils ont besoin de communiquer ?

Nous avons ici définis un nouveau besoin de composant logiciel :

- Auquel les microservices se déclarent lors de leur initialisation.
- Qui enregistre les adresses de chaque microservices.
- Qui route les appels vers le bon destinataire.

Effectuer des appels asynchrones

Ces appels se font en utilisant un bus d'échange de message. Chaque service envoie des messages sur le bus qui seront consommés ensuite par d'autres services. L'émetteur publie sur le bus. Le ou les destinataires qui se sont abonnés à ce type d'évènements sur le bus sont notifiés et peuvent alors récupérer des données. L'émetteur de l'évènement n'a pas de connaissances des clients qui s'abonnent. Ce type d'appels permet de moins coupler les services entre eux. En effet chaque service n'a qu'un seul correspondant : le bus de messages !

Cette façon de communiquer entre applications nous rappelle les fameux ESB (Enterprise Service Bus) adoptés massivement par les SI il y a quelques années. Ces composants vendus par des sociétés tierces ont souvent causé bien des problèmes aux DSI. Mais là où les ESB étaient « intelligents » (transformation, orchestration, routage) un bus de service sert uniquement de canal de transport. Son seul rôle est de stocker l'information en attendant qu'un service tiers la récupère.

La deuxième architecture avec le bus de messages a beaucoup d'avantages mais est plus lourde à mettre en œuvre.

Cette implémentation permet une plus grande flexibilité que les communications synchrones classiques :

- Découplage entre les services : le client envoie sa requête sur un canal sans connaître le service qui va la traiter.
- Message tampon : les messages sont placés dans des files d'attente et seront traités de façon asynchrone par le service même s'il n'est pas disponible au moment de l'envoi du message.
- Communications interprocessus explicites : il n'y a pas de différences entre un appel à un service local ou à distance.
- L'asynchronisme rend plus robuste et plus tolérant le système. Le bus permet de scaler horizontalement tout type de services simplement et efficacement.

Nous venons de traiter deux types d'architectures microservices. La première avec communication par service REST est certainement la plus simple à mettre en œuvre. Elle est efficace et rapide à implémenter lorsqu'il s'agit de migrer depuis une application monolithique.

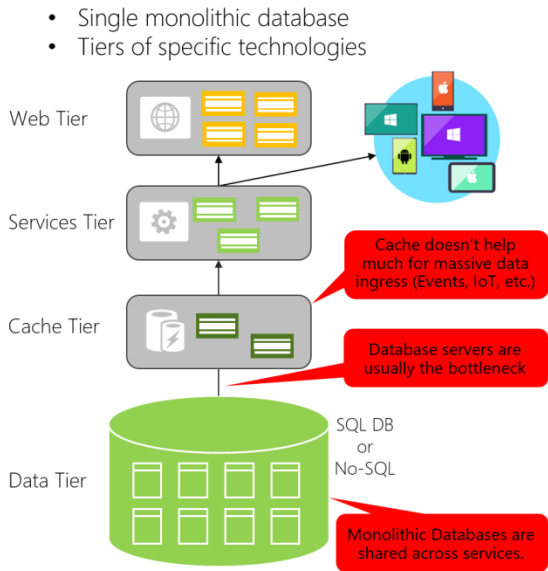
Il est par ailleurs tout à fait envisageable de mixer les deux types d'architecture afin d'obtenir une communication hybride. Ceci peut par exemple être envisagé lors d'une migration progressive vers du tout asynchrone.

Dans tous les cas, quel que soit le modèle choisi, il est fondamental de documenter, via des diagrammes de séquences par exemple, les dépendances entre services, car c'est ici que se situe la complexité de votre système d'informations !

3.2.2.3. Gestion des données

Les services partagent-ils la même source de données ou faut-il privilégier l'approche un service, un datastore ?

Data in Traditional approach



Data in Microservices approach

- Graph of interconnected microservices
- State typically scoped to the microservice
- Remote Storage for cold data

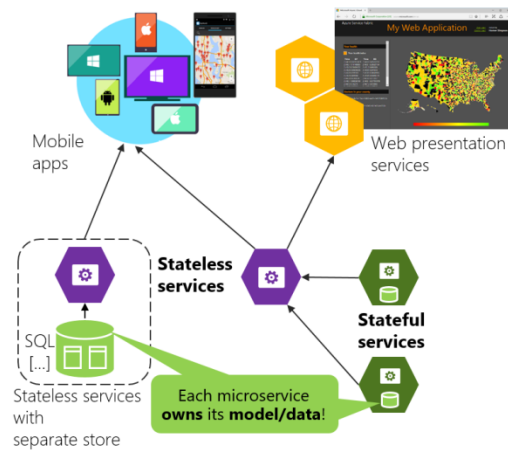
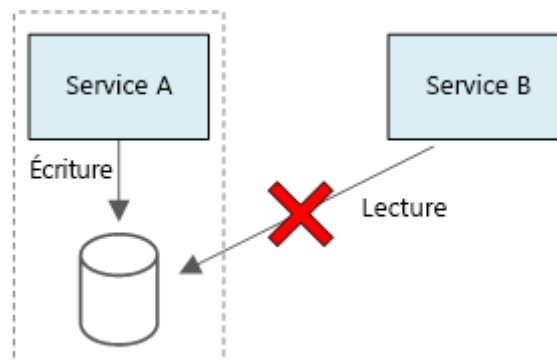


Figure 4-7. Comparaison de la souveraineté des données : base de données monolithique et microservices

Chaque microservice est responsable de ces données et de leurs cohérences. C'est une dépendance externe, et elle doit être adaptée en fonction des besoins :

- Base de données relationnelle
- NoSQL (MongoDB par exemple)
- En mémoire



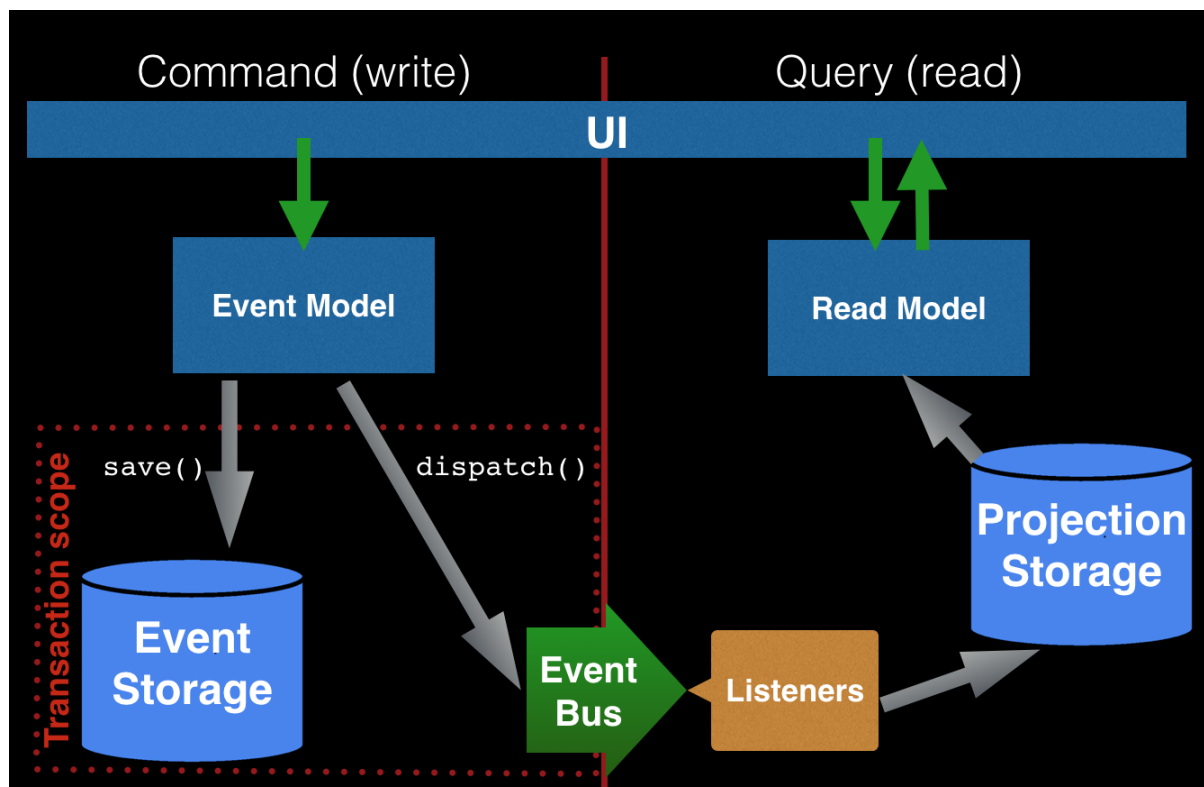
Toutefois, il est tout à fait acceptable que les services partagent le même serveur de base de données physique. Mais chaque service doit être le garant de ces données ! Le problème survient quand les services partagent le même schéma, ou lisent et écrivent sur le même jeu de tables de base de données.

Il n'y a pas d'obligation de provisionner une base de données pour chaque service. Avec un SGBDR nous avons plusieurs possibilités :

- Chaque service à son jeu de tables attribués et ne peut effectuer des opérations CRUD que sur ces tables. Un seul microservice doit avoir accès à une table de la base de données. Il est donc interdit à un service B de réaliser une opération CRUD sur les données dont est responsable le service A.
- Un service pour un schéma de la base.
- Un service : une base de données.

Puisqu'on évoque le principe de « separation of concern » au niveau entités, on peut aller plus loin encore dans cette démarche. Le CQRS (Command and Query Responsibility Segregation) est une architecture qui permet de séparer la lecture (Query) de l'écriture (Command).

Le pattern CQRS consiste à séparer le modèle d'écriture du ou des modèles de lecture. Cela répond à un constat simple : les applications ont des besoins différents en écriture et en lecture.



- La lecture doit être performante. Elle permet d'accéder à des données agrégées et filtrées nécessitant une dénormalisation.
- L'écriture nécessite des performances moindres car souvent moins importante en termes de volumétrie. Par contre, elle gère des règles métiers complexes et la cohérence entre les données.

Cette approche peut être intéressante si une opérations d'écriture nécessite des traitement couteux sur les données car elle permet de ne pas impacter les clients en lecture.

3.2.2.4. Gestion des logs et monitoring

Comment gérer la collecte de log et le monitoring dans un système éclaté, distribué et auto scalable ?

Le besoin de logs et de monitoring est un enjeu majeur, dans cet environnement plus qu'ailleurs. Nous avons besoins d'outils permettant de :

- Récolter les logs applicatifs
- Récolter les logs systèmes et réseaux
- Les présenter de façon lisible et structurée
- Récolter les métriques de performances systèmes et réseaux
- Les présenter sous forme de graphique
- Surveiller les métriques en temps réel.

3.2.2.5. Gestion des pannes et résilience

Comment garantir à nos clients une SLA optimale compte tenu du caractère fortement distribué de notre SI ?

Il ne faut pas perdre de vue que résister à la panne ne veut pas dire infaillible. Quand un service tombe, il faut un mécanisme pour identifier cette panne et relancer le service en question.

Cette résilience des architectures microservices passe donc par :

- Un mécanisme de surveillance.
- Un démarrage et une configuration des serveurs automatisés.
- Un déploiement des services automatisé.

Notre architecture mettant en œuvre un grand nombre de composants communiquant par HTTP le nombre d'erreurs est forcément plus important que dans un cadre d'exécution plus classique et une communication inter processus.

Les besoins de notre système sont :

- Détecter quand un service est off pour ne pas que le client tente de le joindre inutilement.
- Auto guérison des services : le cloud favorise la gestion de ce besoin par la mise en œuvre de :
 - Scalabilité horizontale : duplication automatique des composants logiciels.
 - Scalabilité verticale : augmentation de la puissance de la plateforme hôte (CPU / RAM / espace disque).

Au niveau logiciel, on peut également agir en mettant en œuvre un pattern appelé le circuit breaker. Il permet de contrôler la collaboration entre différents services afin d'offrir une grande tolérance à la latence et à l'échec. Pour cela, en fonction d'un certain nombre de critères d'erreur (timeout, nombre d'erreurs, élément dans la réponse), ce pattern permet de désactiver l'envoi de requêtes au service

appelé et de renvoyer plus rapidement une réponse alternative de repli (fallback), aussi appelé graceful degradation.

La gestion des erreurs dans ce type de système est tellement importante que Netflix à prévu un composant chargé de débrancher des services aléatoirement pour s'assurer de la résilience de son système : c'est le célèbre Chaos Monkey.

« Le but de cet outil est de simuler des pannes en environnement réel et de vérifier que le système informatique continue à fonctionner. »
https://fr.wikipedia.org/wiki/Chaos_Monkey

On s'assure ainsi d'avoir anticipé correctement la survenue de ce type d'incidents en mettant en place une architecture suffisamment redondante pour qu'une panne de serveurs n'affecte d'aucune façon les millions d'utilisateurs de Netflix.

3.2.2.6. Gestion des transactions

Les microservices peuvent-ils partager une transactions ?

Si une opération de mon SI nécessite deux microservices, comment font-ils pour se partager la transaction et garantir la cohérence du système en cas d'erreur ? Il faudrait pouvoir faire un rollback en cas d'échec et un commit en cas de succès mais sur les deux services via une même transaction.

Cela ne posait pas de problème avec SOAP et WCF (voir le standard WS-AT). Ici, avec REST ce n'est tout simplement pas possible !

Il faut donc gérer ce problème de façon logiciel.

3.2.3. Application Lifecycle Management

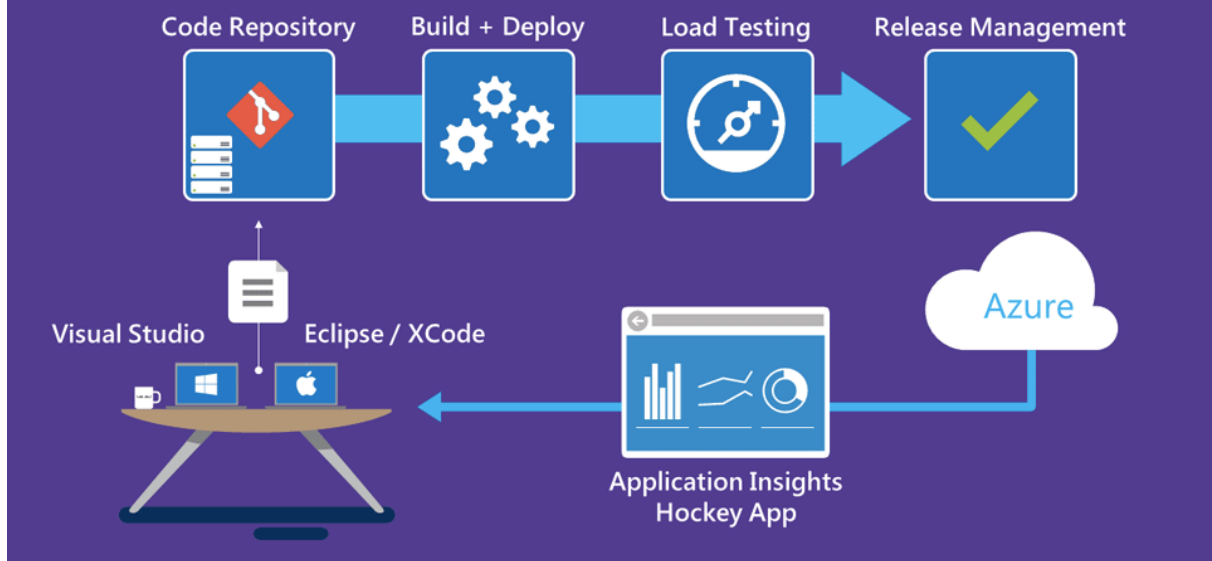
Le but d'un microservice étant d'être isolé, facilement testable et déployable, un critère important de cette mise en œuvre est l'utilisation d'un outil de build et de déploiement continu (continuous intégration / continuous déploiement).

Des solutions existent sur le marché comme TeamCity ou Jenkins. Compte tenu de notre environnement Microsoft nous utiliserons Visual Studio Team Services, outils qui réponds parfaitement à notre besoin.

VSTS permet :

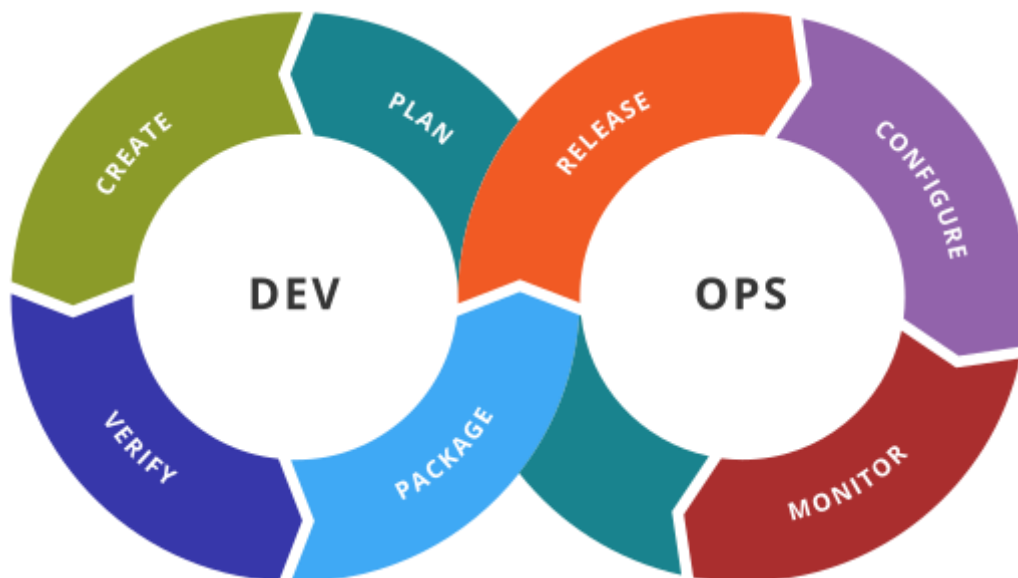
- D'avoir une forge logicielle avec un outils de gestion de version : git, tfvc.
- De configurer des processus de build avec exécution de tests automatisés
- De configurer des processus de releases : déploiement sur des serveurs ou dans le Cloud.

Visual Studio Team Services



Au-delà de l'aspect CI/CD VSTS permet également de gérer un projet en méthode agile avec la gestion des itérations de développement (sprints), les work items, backlog etc ...

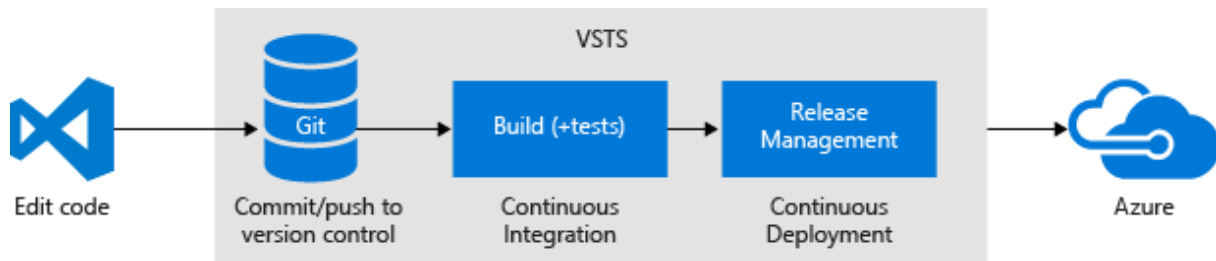
Une maturité DevOps est obligatoire pour la mise en œuvre de notre architecture. En effet, nos services seront nombreux et doivent pouvoir être déployé rapidement suite à une évolution. Sans une plateforme DevOps d'industrialisation logicielle pour nous aider, la tâche s'annonce plus que compliqué.



3.2.3.1. Intégration

La partie intégration est assurée par :

- Un outil de gestion de version : Git qui n'est plus à présenter ou TFVC, l'outil de gestion de version Microsoft, en fin de vie. La firme ne conseille plus son utilisation. A noter que VSTS peut également récupérer les sources d'un dépôt disant comme GitHub.
- Un outil de Builds, intégré à VSTS.
-



Les builds sont configurables, s'exécutent sur une branche du repository git et permettent de réaliser un certain nombre d'actions :

- Télécharger les dépendances Nuget du projet.
- Compiler la solution.
- Exécuter les tests automatisés.
- Exécuter des commandes Windows.
- Pousser l'artifact généré .
- Envoyer un rapport de build par mail.
- Lancer un tests Sonar.
-

De la façon la plus concise possible, sur un projet de type .Net core la build se déroulera en 4 commandes :

- dotnet restore
- dotnetbuild
- dotnettests
- dotnetpublish

3.2.3.2. Tests

Pour nos services, différents types de tests sont à mettre en œuvre pour garantir la stabilité du code source et éviter les régressions :

Tests unitaires

Chaque test valide le bon fonctionnement d'une méthode bien précise de notre service, l'utilisation d'un framework de mock (moq, rhinoMock) est indispensable pour garantir le caractère « unitaire » de nos tests. Un framework de mock permet de simuler le fonctionnement d'une méthode.

Par exemple, nous testons la méthode B. Dans l'implémentation de B il y a un appel à la méthode A qui retourne un booléen. Alors pour notre test de B on précisera en amont que notre méthode A renverra systématiquement true si elle est appelée. Pendant notre test si A() est appelé, on est assuré d'avoir le résultat true, et on peut tester notre méthode B indépendamment de A. Microsoft propose son propre framework de mocking : Microsoft.Fakes.

A noter que la mise en œuvre d'une stratégie Test Driven Development (TDD) peut aider à garder une couverture de tests satisfaisante.

Voir <http://igm.univ-mlv.fr/~dr/XPOSE2009/TDD/pagesHTML/PresentationTDD.html>

Tests d'intégration

Chaque test valide le bon fonctionnement d'une méthode de notre service, de l'appel jusque la base de données. Le test balaie donc la totalité de notre méthode. Par exemple avec l'appel GET <http://www.demo.com/clients/2>, on pourra valider qu'en sortie on obtient un client provenant de la base de données dont l'identifiant est 2.

Tests de qualité

Il est possible d'intégrer un outil comme Sonar pour la qualimétrie logicielle et l'analyse du code :

- Respect des normes de développement
- Complexité cyclomatique
- Présence des commentaires
- Duplication de code
- Evaluation de la couverture de code



Tests statiques

On peut utiliser le module de code review de VSTS pour la relecture de code entre développeurs. Les tests statiques sont effectués par les développeurs avant de pousser leurs modifications sur la forge. Les tests unitaires, d'intégration et de qualité quant à eux pourront être exécutés de manière automatisée par notre outil DevOps pendant une build. En cas d'erreur sur un des tests, l'archivage sera rejeté pour être corrigé par le développeur.

3.2.3.3. Déploiements

La partie Déploiements de notre solution est géré par les « Releases » de VSTS. Les releases permettent à partir d'une build de déployer nos dll dans un environnement cible, serveurs interne ou environnement cloud.

L'hébergement de notre architecture microservices peut se faire dans un environnement Cloud pour répondre facilement aux besoins de disponibilités et de scalabilité des applications. Des solutions existent telles que Azure (Microsoft), AWS (Amazon Web Services), Google Cloud Platform.

Ici nous nous intéresserons uniquement à Azure et aux différentes solutions offertes par cette plateforme pour répondre à notre besoin.

Les releases permettent en plus du déploiement d'effectuer des opérations comme :

- Jouer des scripts SQL
- Déplacer des fichiers (xml, json)
- Exécuter des commandes PowerShell, FTP
- ...

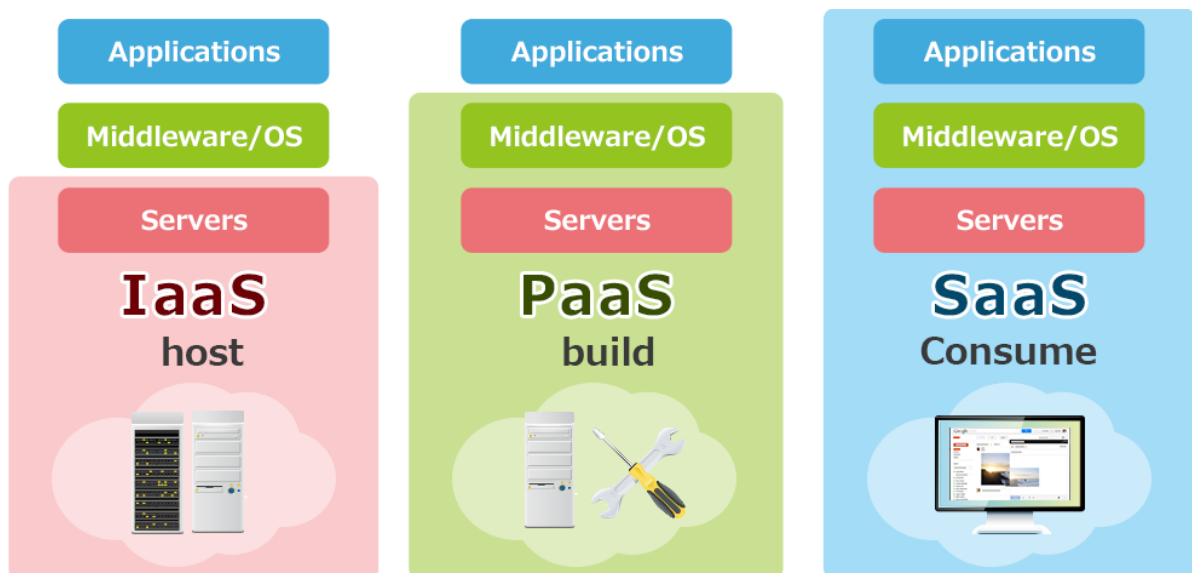
A noter que des outils externes à Microsoft s'interface très bien avec VSTS, au niveau des builds comme des releases pour exécuter diverses tâches. Je pense notamment à Octopus, Sonar, Docker, WebPack ... Un système de plugin est même mis en œuvre pour pouvoir ajouter des fonctionnalités à VSTS.

Nous verrons dans le chapitre 4.5 quelle stratégie nous pouvons mettre en œuvre pour ne pas entraîner d'interruptions de service pendant le déploiement et comment effectuer un rollback en cas de problèmes lors de la mise en production.

3.3. Hébergement dans Azure

3.3.1. Présentation générale

Azure est une plateforme Cloud d'hébergement d'applications, de données et de services créés en 2008 par Microsoft. Les solutions proposées par Azure sont nombreuses. Ils se déclinent en trois principales catégories :



L'IaaS (*Infrastructure as a Service*) est une externalisation de l'infrastructure informatique matérielle. L'offre gère pour nous :

- L'installation des serveurs fichiers
- Les réseaux
- Le stockage de données

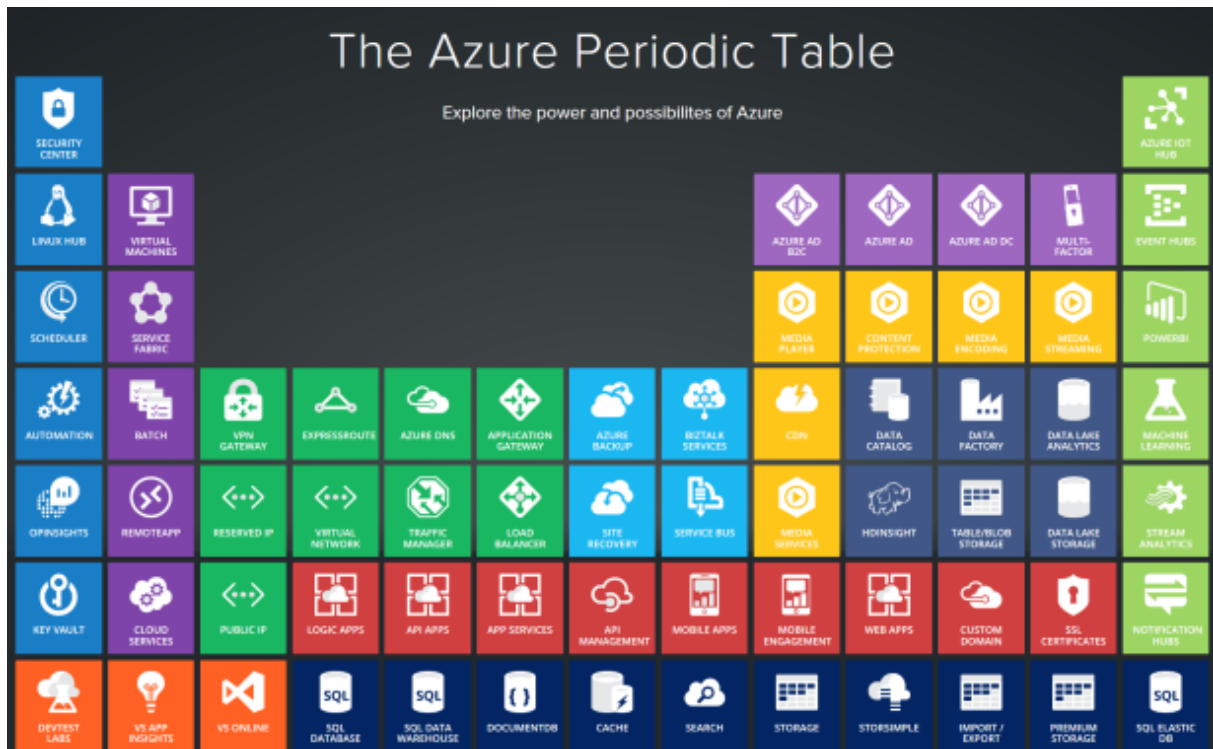
Le PaaS (*Platform as a Service*) est un IaaS plus poussé. Il consiste à sous-traiter non seulement l'infrastructure matérielle mais également vos applications *middleware* :

- Systèmes d'exploitation
- Base de données
- Serveurs web

A nous ensuite d'y ajouter nos applications et services.

Le SaaS (*Software as a Service*) est la version la plus aboutie et la plus complète de l'externalisation de composants du système d'information. Le fournisseur gère, pour nous :

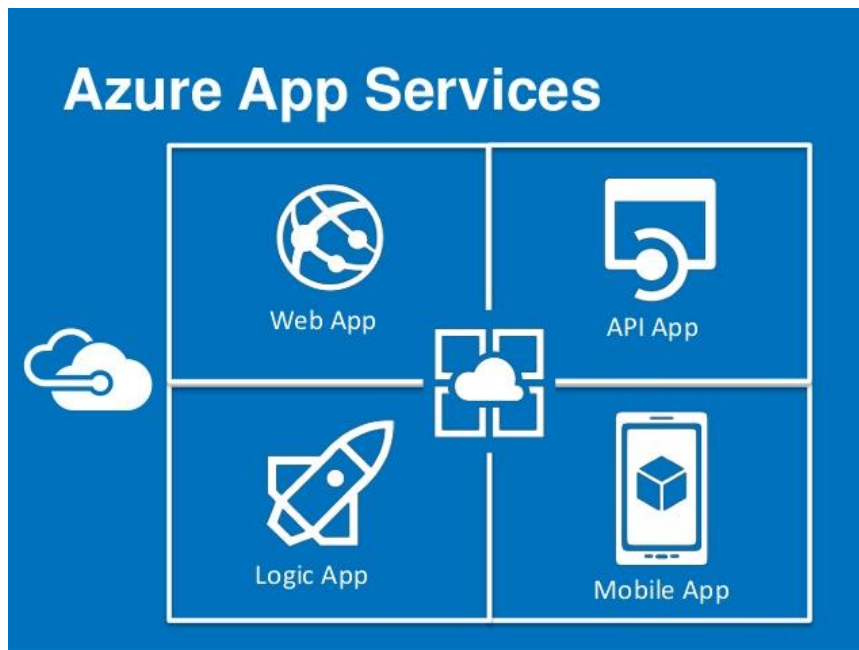
- L'installation
- La configuration
- Le fonctionnement
- La maintenance



Pour l'hébergement d'applications de type microservices dans Azure nous disposons de plusieurs possibilités avec des offre de type PAAS :

3.3.2. Web App

Les web applications constituent le moyen le plus simple d'héberger une application dans Azure. Il s'agit d'une solution PAAS qui fait partie du groupe de fonctionnalités Azure App Services.



Web App est un service pour l'hébergement d'applications web, d'API REST et de backends mobiles. Vous pouvez y déployer des applications développées en .NET, .NET Core, Java, Ruby, Node.js, PHP ou Python. Les Web App sont déployés sur des machines virtuelles Windows ou Linux.

Il est également possible d'y déployer des applications s'exécutant dans un conteneur Docker.

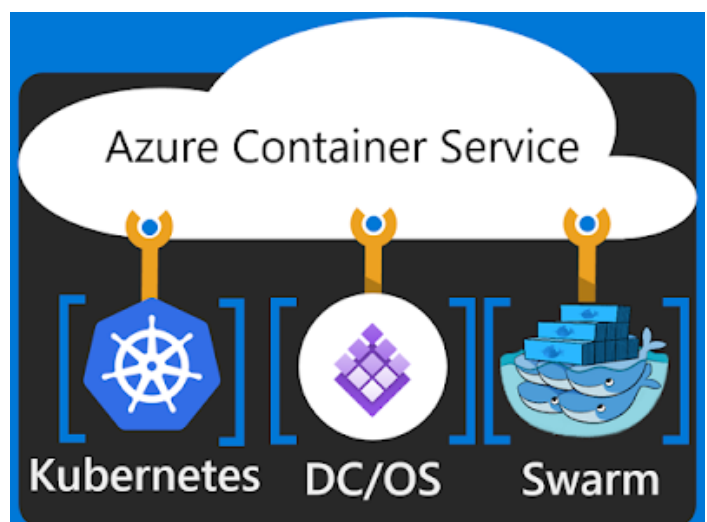
3.3.3. Azure Container Service (ACS)

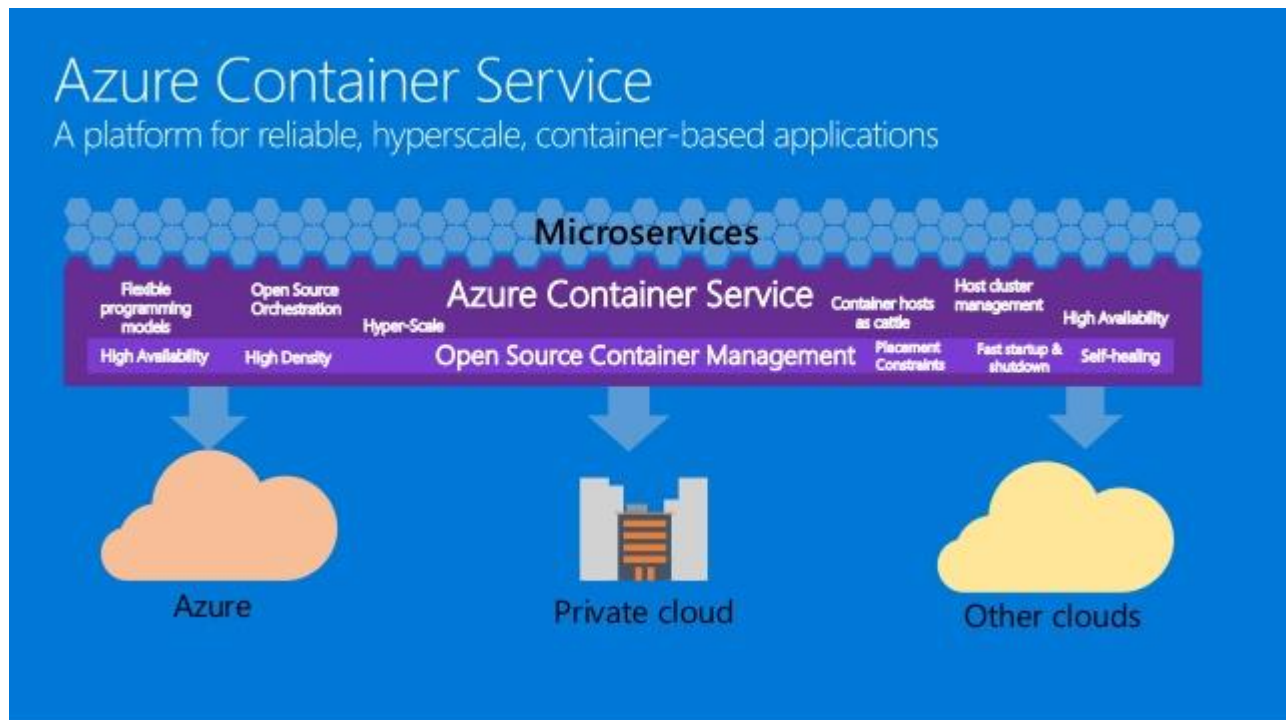
On passe à une solution PAAS bien plus complète que les « simples » WebApp.

ACS est une plateforme de conteneur pouvant être orchestré par :

- DC/OS
- Docker Swarm
- Kubernetes (l'orchestrateur

Docker de Google)





Azure Container Service est une solution permettant de déployer, provisionner et manager des conteneur Docker.

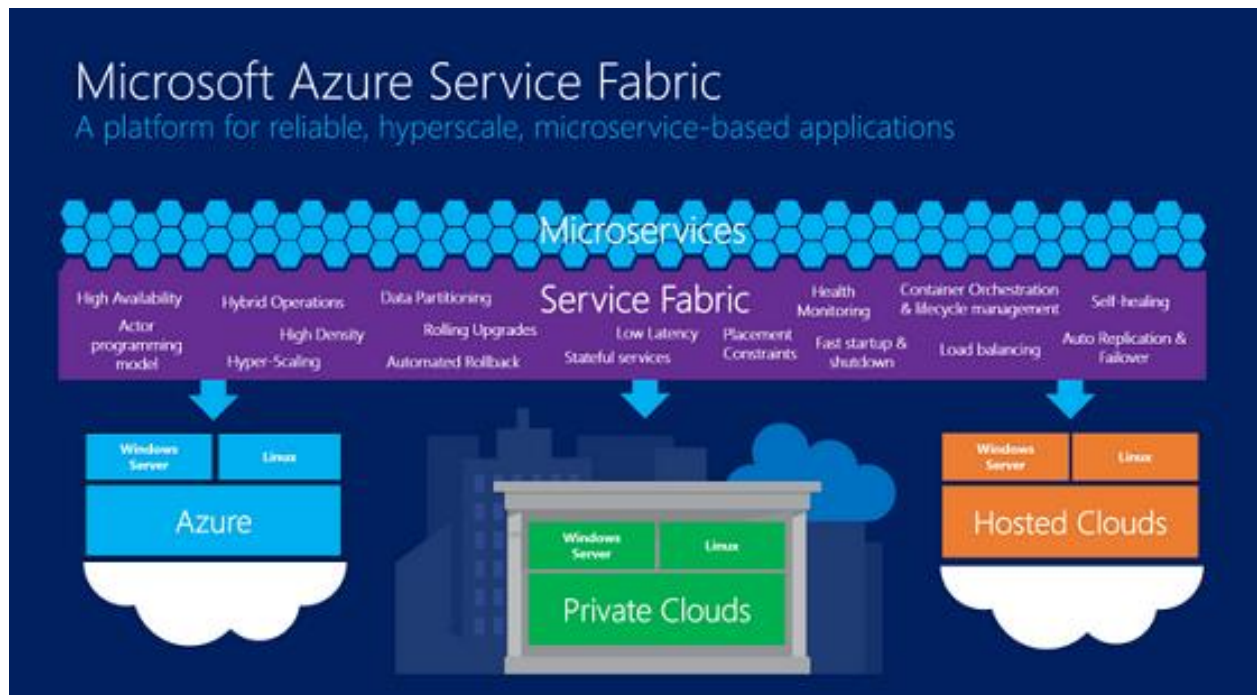
La technologie d'un conteneur virtualise le système d'exploitation par rapport aux applications. Par rapport aux machines virtuelles, les conteneurs présentent les avantages suivants :

- Ils sont bien plus légers. Un conteneur ne contient que l'application à déployer et ces dépendances.
- Démarrage rapide : comme les conteneurs n'ont pas besoin d'initialiser l'intégralité d'un système d'exploitation, ils peuvent démarrer beaucoup plus rapidement, généralement en quelques secondes.
- Portabilité : une image d'application en conteneur peut être portée de manière à s'exécuter dans le cloud ou sur site, à l'intérieur de machines virtuelles ou directement sur des machines physiques.

L'utilisation de cette plateforme me semble intéressent pour un SI qui maitrise déjà docker et un orchestrateur et qui veut migrer vers le Cloud. Historiquement Docker et les orchestrateurs de conteneur adresse plutôt des technologies open source mais se sont tournés récemment vers Microsoft grâce à .Net Core.

La troisième option est le concurrent direct d'ACS.

3.3.4. Azure Service Fabric



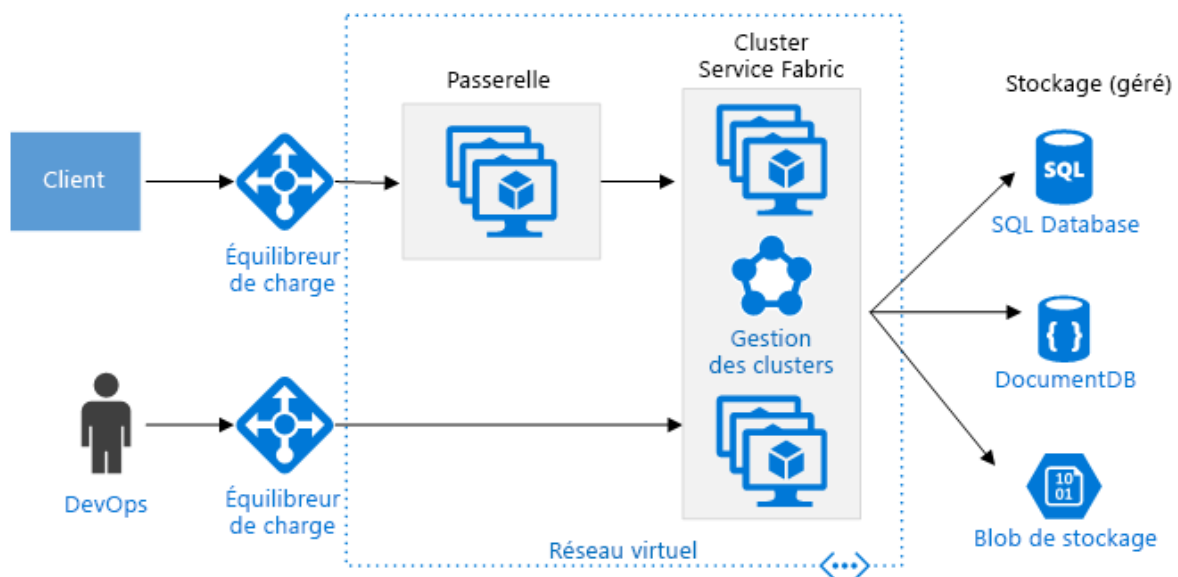
Azure Service Fabric est la plateforme de microservices signée Microsoft. Elle peut être déployée OnPrem sur des serveurs Windows Server 2016 ou Linux, dans le Cloud sous Azure, mais également dans n'importe quel autre Cloud.

La plateforme est mise à disposition de tous depuis 2015 mais Microsoft l'éprouve depuis plusieurs années. En effet plusieurs services Azure utilisent cette infrastructure :

- Skype pour les entreprises
- DocumentDB
- Event Hubs
- Azure SQL Database (qui héberge plus de 1,4 million de bases de données clients)
- Cortana qui peut traiter plus de 500 millions de requêtes par seconde.

Source MSDN : « Azure Service Fabric est un orchestrateur de services sur un cluster de machines. Azure Service Fabric est une plateforme de systèmes distribués qui facilite le packaging, le déploiement et la gestion de conteneurs et de microservices évolutifs et fiables. Les développeurs et administrateurs sont en mesure d'éviter les problèmes d'infrastructure complexes et peuvent se concentrer sur l'implémentation de charges de travail stratégiques et exigeantes, évolutives, fiables et faciles à gérer. »

Autrement dit, il s'agit d'une plateforme PaaS associée à un système d'orchestration des services, façon Kubernetes.



Azure Service Fabric propose :

- Une gestion de la haute disponibilité des services.
- Un modèle de programmation simple.
- Une scalabilité des services à grande échelle.
- Un partitionnement des données.
- Une gestion fine des mises à jour que ce soit en upgrade ou en downgrade.
- Un monitoring simple de la santé de vos services.
- Un démarrage et un arrêt rapide des services.
- La gestion du load balancing des services.
- Une gestion de récupération automatique des services.
- Un système de réplication et de gestion de failover.

Les différents types d'applicatifs pouvant être exécutés dans Service Fabric sont :

- Stateless service : Un service au sens courant qui ne maintiens pas d'état entre les appels.
- Stateful service : Un service capable de conserver des informations grâce aux ReliableDictionary qui garantis une persistance des données entre les différentes instances du service.
- Actor service : Par exemple, un panier d'un utilisateur sur un site de e-commerce peut-être un actor service.
- Guest exécutable : Permet d'exécuter une application exe dans Service Fabric
- Container : Exécutions d'un conteneur docker dans ASF : disponible récemment.

Ces différents types de services peuvent etre développé en utilisant .Net ou .Net Core.

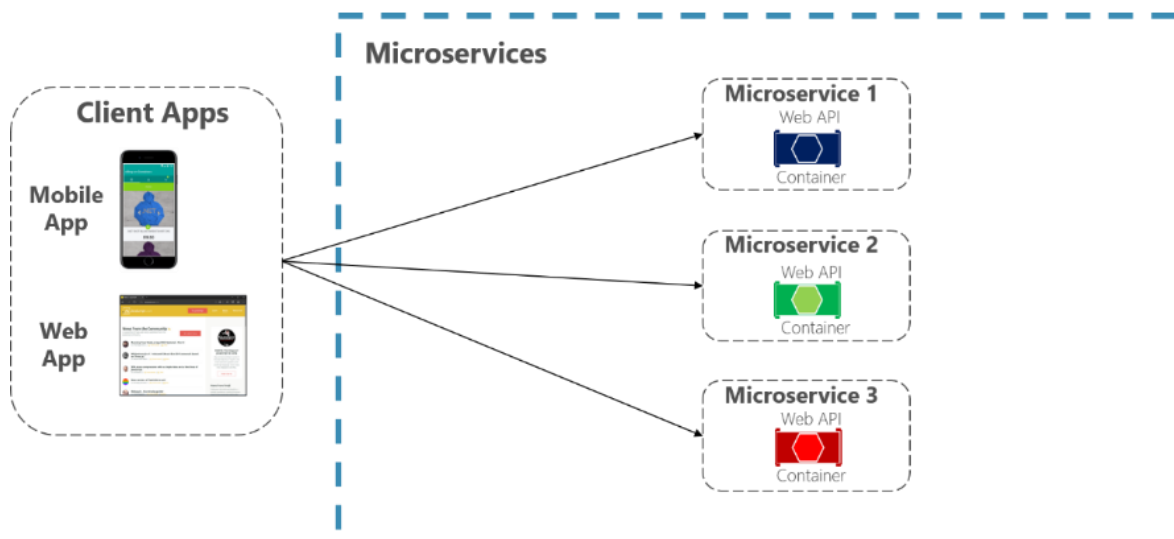
Nous allons maintenant reprendre point par point les problématiques relevés pour l'implémentation d'une architecture microservices dans le chapitre 3.2.2 et détailler comment Azure Service Fabric réponds à chacun d'entre eux.

3.3.4.1. Communication clients / services

Nous avons deux possibilités pour qu'un client (web, windows, mobile) puissent échanger avec un service :

- Communication directe

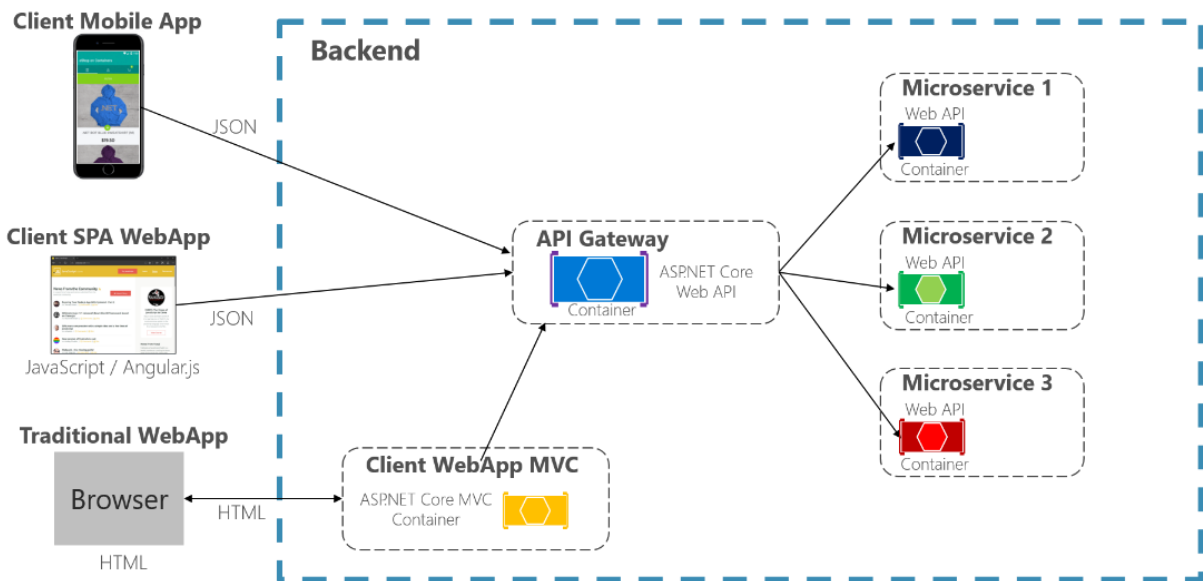
Direct Client-To-Microservice communication Architecture



- Utilisation d'un composant intermédiaire : un API Gateway

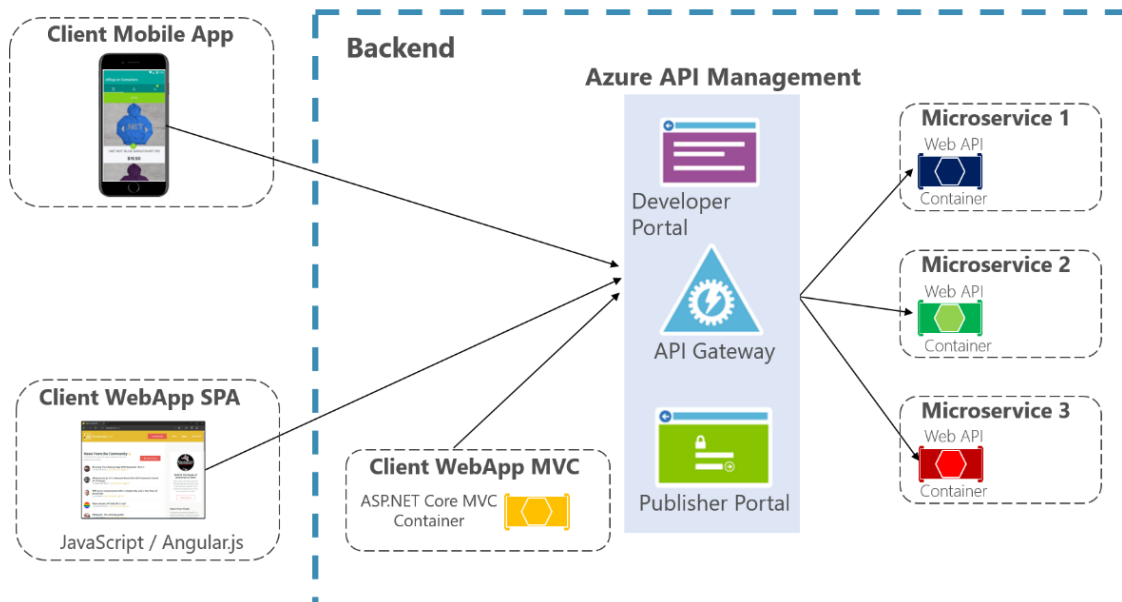
Nous pouvons développ   notre propre API Gateway :

Using the **API Gateway Service**



Ou utiliser un API Gateway de Type Azure API Management

API Gateway with Azure API Management Architecture



Cet outil permet de configurer simplement un API Gateway dans Azure. Il n'y a pas de d  veloppement suppl  mentaire    produire.

APIM est con  ue pour :

- Gérer les API complexes avec des règles de routage,
- Un contrôle d'accès (Authentification)
- Une surveillance
- Une journalisation des événements
- Une mise en cache des réponses

Attention en contrepartie notre point d'accès unique à notre API est par définition un point de défaillance important de notre système. S'il tombe, il n'y a plus de communication possible. Du côté du trafic réseaux il peut également devenir un goulot d'étranglement s'il ne s'adapte pas à la charge.

3-3-4-2. Communication entre services

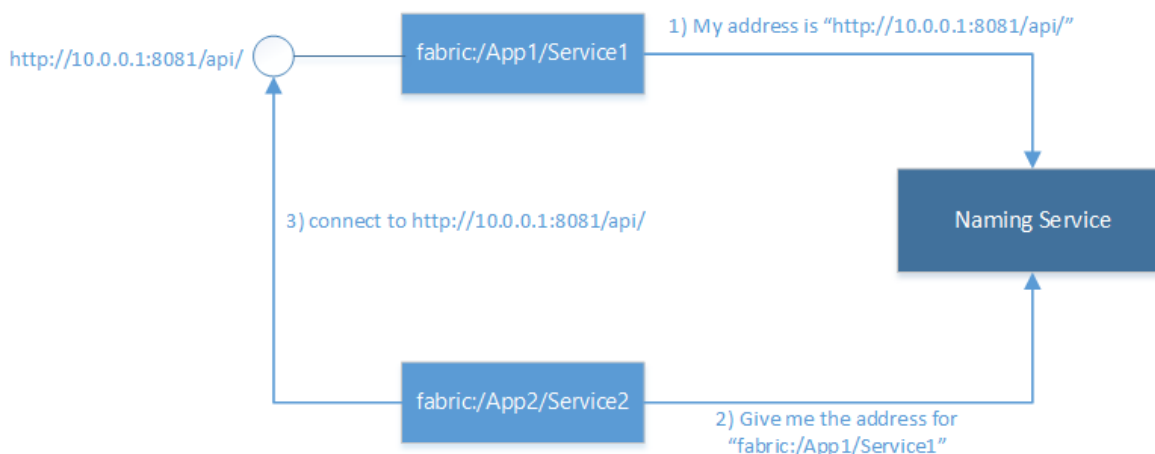
Pour la communication entre services nous avons listé deux possibilités :

Echange synchrone via commande HTTP.

Attention ici il s'agit bien de l'appel qui est synchrone dans le sens où le contact entre les deux services est direct. Au niveau applicatif l'appel peut être asynchrone dans le sens où le client peut faire un traitement en attendant une réponse du service.

Si la communication entre service est directe, cela implique que les services sachent à quelle adresse ils peuvent s'appeler. Pour cela, Azure Service Fabric nous propose une solution de Reverse Proxy.

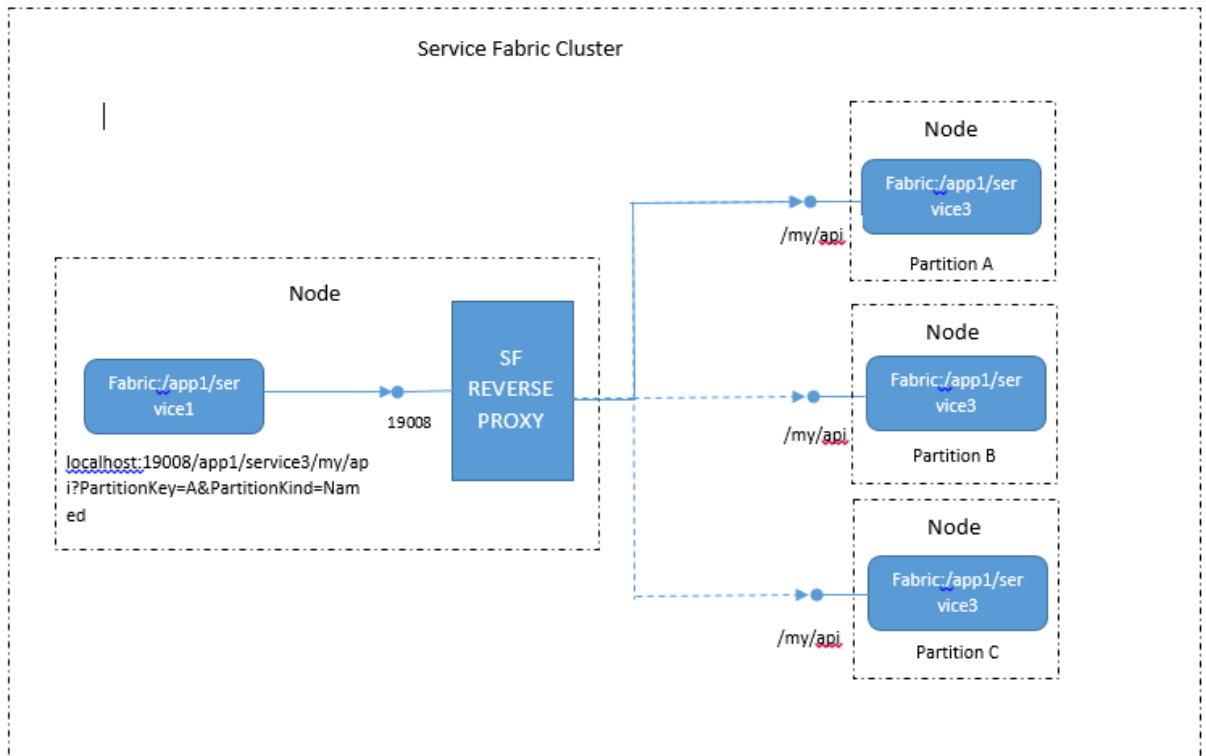
Au démarrage d'un service celui-ci vient s'identifier auprès du service d'attribution de noms d'ASF. Celui-ci tient une table qui mappe les instances de service nommées sur les adresses de point de terminaison qu'elles écoutent.



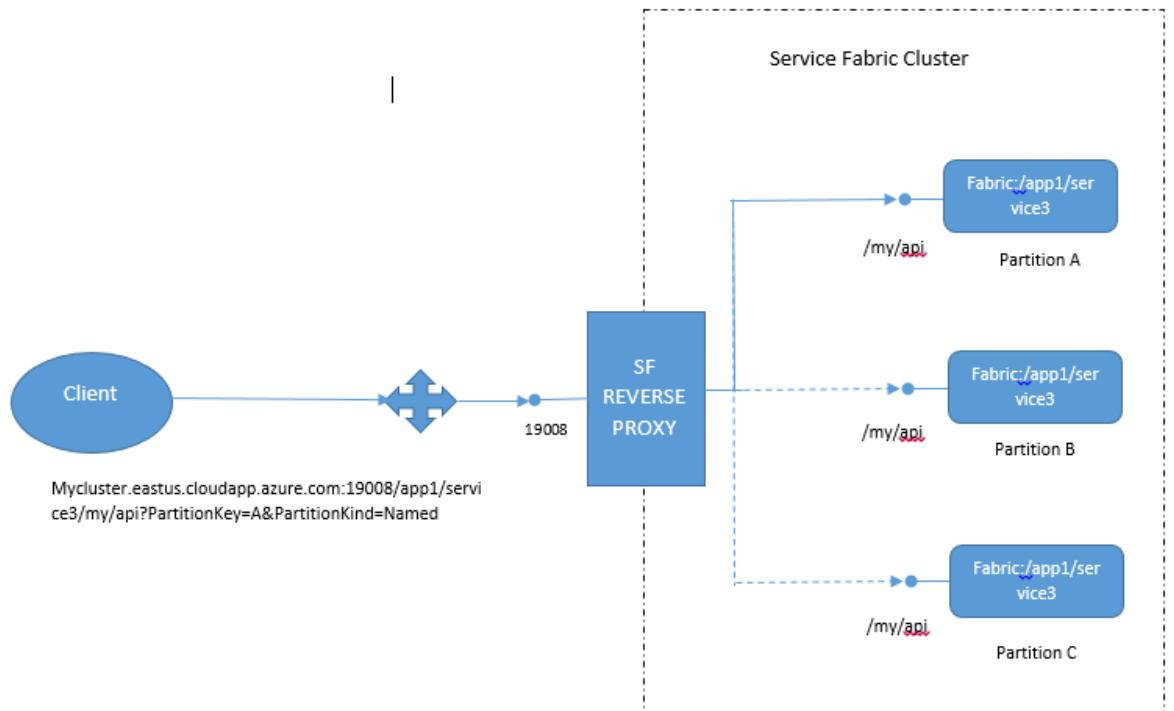
La résolution et la connexion aux services impliquent l'exécution des étapes suivantes en boucle :

- Résoudre : obtenir le point de terminaison publié par un service à partir du Service d'attribution de noms.
- Connecter : se connecter au service sur le point de terminaison en question.
- Si la tentative de connexion échoue, les étapes précédentes de résolution et de connexion doivent être réessayées, et ce cycle se répète jusqu'à ce que la connexion aboutisse (Pattern circuit breaker)

Appel entre service :



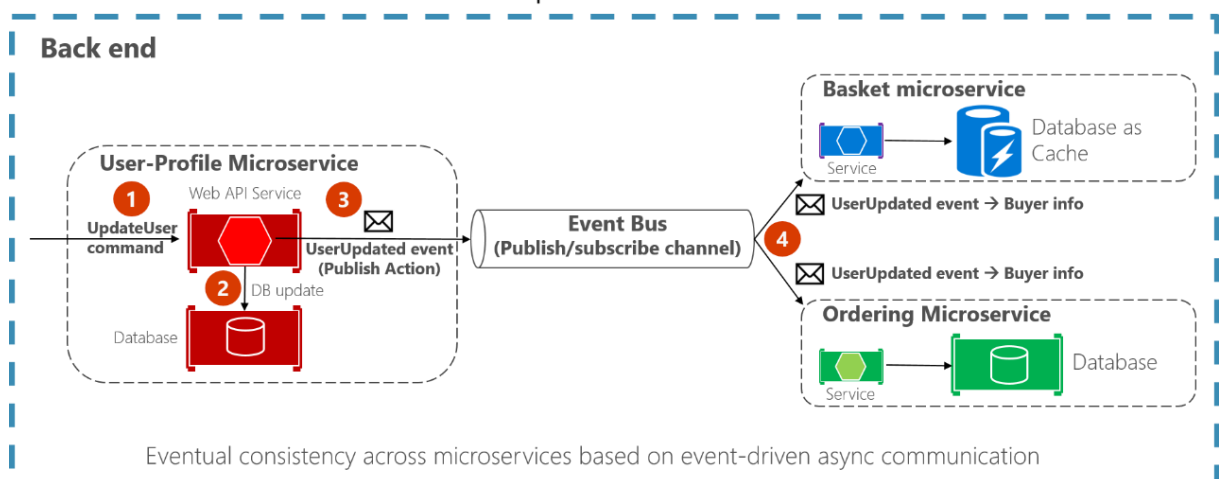
Appel depuis l'extérieur :



- Echange asynchrone via un bus d'évènement basé sur le protocole AMQP.

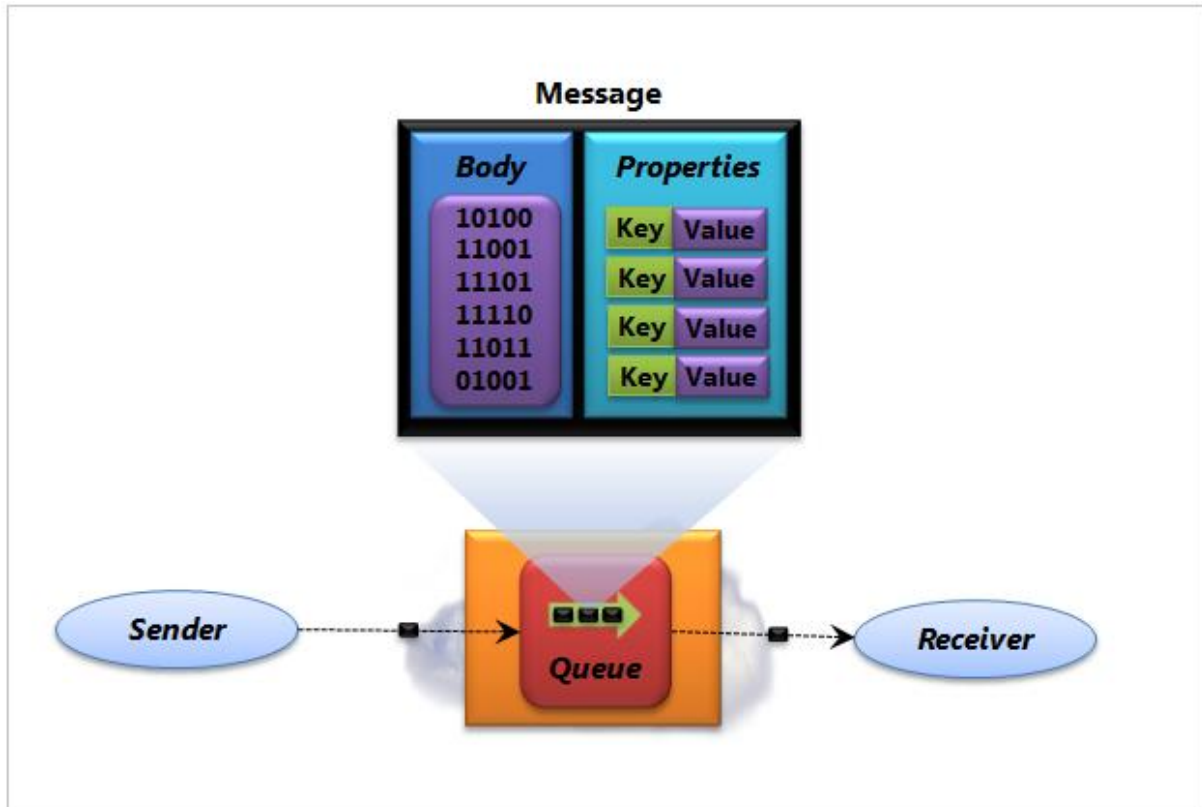
Asynchronous event-driven communication

Multiple receivers



Si nos besoins nous conduisent vers une communication asynchrone, nous pouvons utiliser Azure Service Bus. Il y a alors deux cas d'utilisation.

- Les **files d'attente** permettent la communication unidirectionnelle. Chacune agit comme un intermédiaire (ou broker) qui stocke les messages envoyés jusqu'à leur réception. Chaque message est reçu par un destinataire unique.

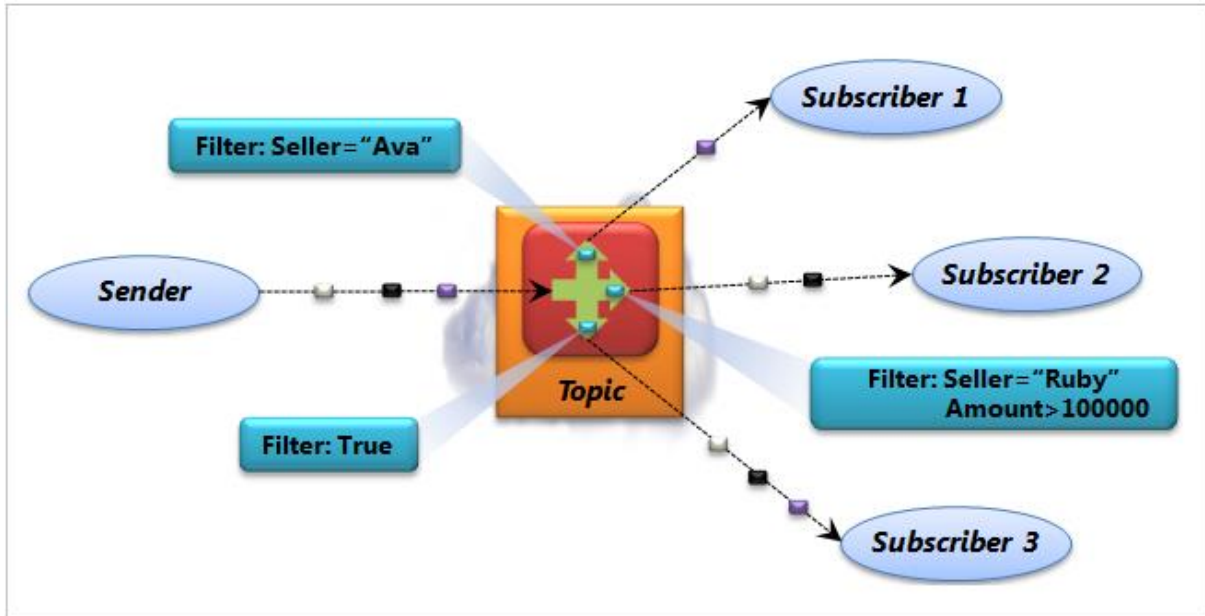


Il s'agit du cas le plus simple pour une communication par message : un émetteur, une file d'attente, un destinataire.

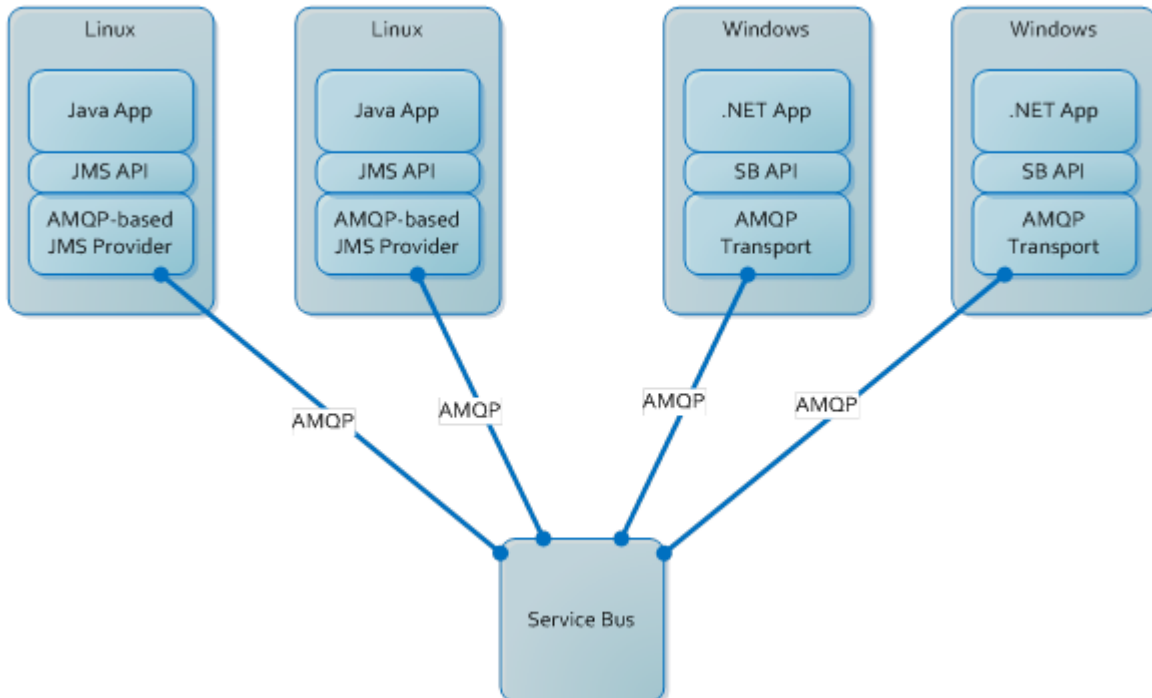
- Les **rubriques** sont assez similaires aux files d'attente.

Les expéditeurs envoient les messages à la rubrique de la même façon qu'ils envoient des messages dans la file d'attente. Ces messages ont le même aspect.

À l'inverse des files d'attente cependant, un message unique envoyé à une rubrique peut être reçu par plusieurs abonnements. Cette approche, communément appelée *publication et abonnement* (ou *pub/sub*), est utile quand plusieurs applications sont intéressées par les mêmes messages.



Les files d'attente et les rubriques dans Azure Service Bus se basent sur le protocole AMQP : Ce protocole de messagerie open standard permet le développement d'applications basées sur des messages à l'aide de composants créés avec plusieurs langages, infrastructures et systèmes d'exploitation.



Qu'ils s'agissent d'une communication synchrone via appel HTTP ou asynchrone avec Azure Service Bus, les deux implémentations sont possibles avec Azure Service Fabric. Les deux solutions ont des avantages et des inconvénients, à chacun de les mesurer en amont du projet.

3.3.4.3. Gestion des données

Pas de remarques particulières à ce niveau. Il est bien sur possible d'utiliser des bases de données existantes. Il est même possible d'héberger tout type de base de données dans Azure si besoin pour répondre aux différentes problématiques Data de vos applicatifs.

- SQL Server
- NoSQL : mongoDB
- MySql

En ce qui concerne la séparation des données par service, ici aussi il n'y a pas de solutions meilleures qu'une autre. Nos options sont :

- Chaque service à son jeu de tables attribués.
- Un service pour un schéma de la base.
- Un service : une base de données.

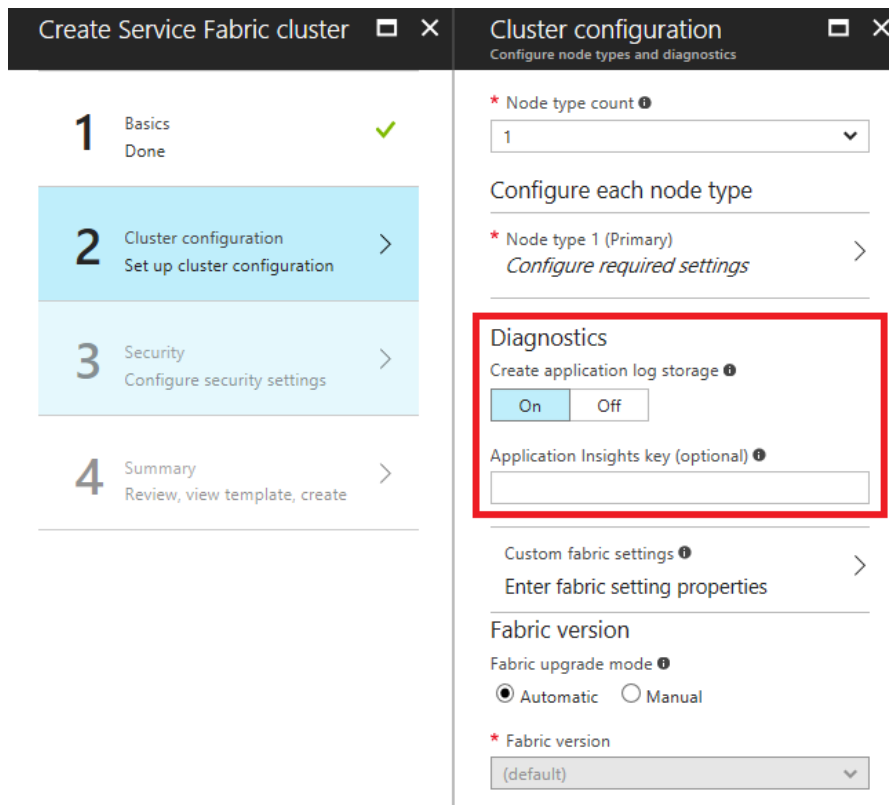
L'architecte positionnera en fonction des contraintes de chaque solution la meilleure pour le système d'informations.

3.3.4.4. Gestion des logs et monitoring

Le monitoring et la gestion des logs sont intégrés à Azure Service Fabric grâce à deux composants Azure :

- Windows Application Diagnostics
- Application Insights

Ces deux modules doivent être activés à la création du cluster Azure Service Fabric et configurés dans la solution Visual Studio.



WAD permet de :

- Détecter et diagnostiquer les problèmes d'infrastructure
- Détecter les problèmes liés à votre application
- Comprendre la consommation des ressources
- Faire le suivi du niveau de performance des applications, des services et de l'infrastructure

Pour cela WAD collecte des « journaux » :

- Evènements de la plateforme ASF
- Intégrité et charge des clusters
- Monitoring et utilisation du Reverse proxy
- Compteur de performance
- Evènement des applications (EventSource)

Application insights quant à lui permet de :

- Profiler nos applications
- Analyser l'utilisation des services
- Obtenir des métriques en temps réel : voir <https://docs.microsoft.com/fr-fr/azure/application-insights/app-insights-metrics-explorer>
- Obtenir une piles d'exception en cas de crash


- Tracer les appels réseaux. Il est alors possible grâce à Metrics Explorer de mettre en forme ces données dans des graphiques personnalisés.

AI nous offre également un langage de requêtage, inspiré de LinQ (Analytics pour Application Insights) pour extraire les données qui nous intéressent :

COMMON QUERIES

USERS


Top 10 countries by traffic in the past 24 hours



[▶ RUN](#)

PERFORMANCE


What are the 50th, 90th, and 95th percentiles of request duration in the past 24 hours?



[▶ RUN](#)

ERRORS


Find what exceptions led to failed requests in the past 24 hours



[▶ RUN](#)

USAGE

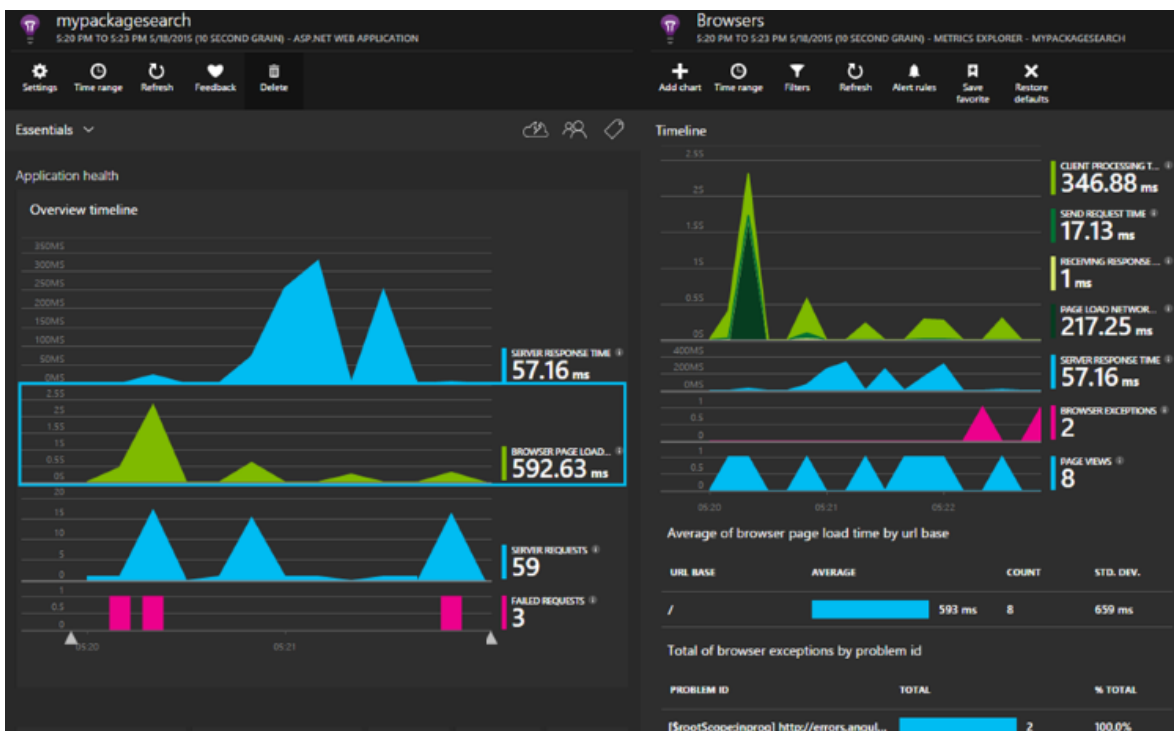
What are the top 10 custom events of your application in the past 24 hours?

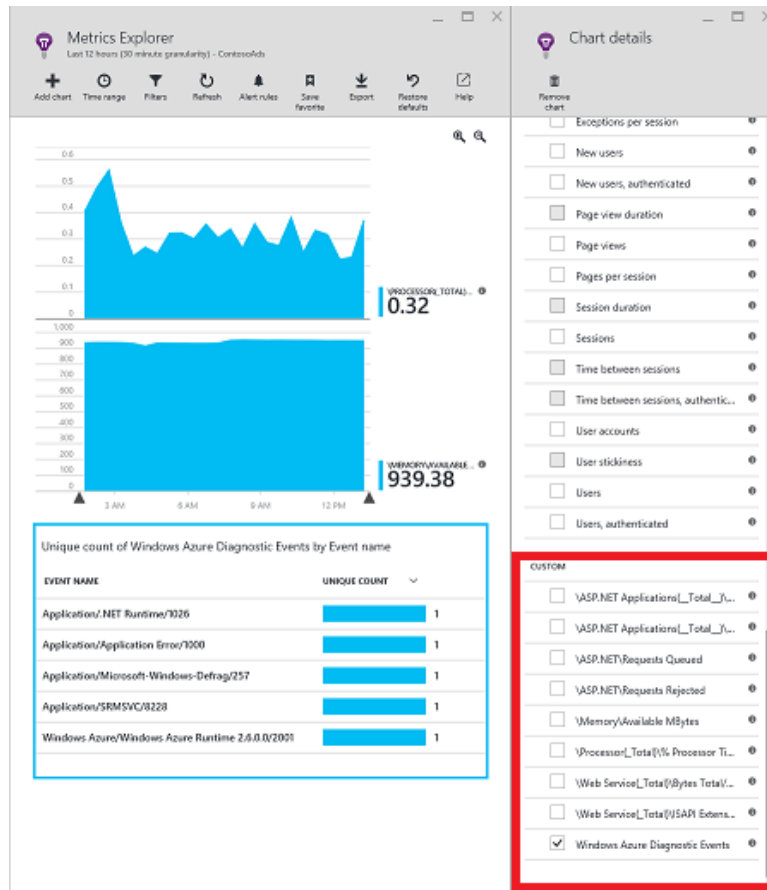


[▶ RUN](#)

```

// Top 10 countries by traffic in the past 24 hours
requests
  | where timestamp > ago(24h)
  | summarize count() by client_CountryOrRegion
  | top 10 by count_
  | render piechart
  
```

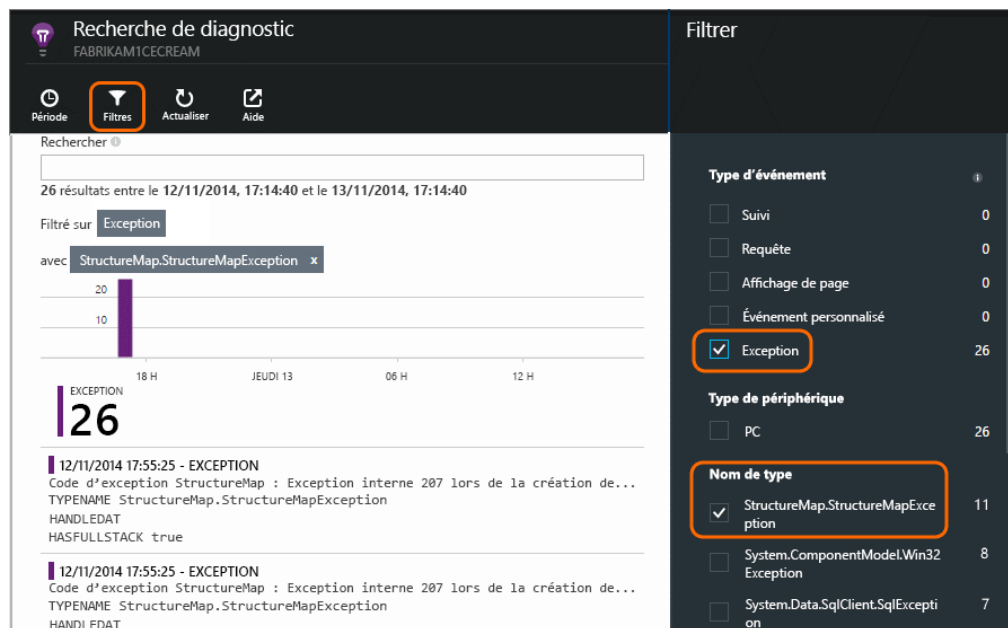




Monitoring dans Application Insights


Insight analyse :


- Les exceptions dans un services :



- Consultations de pages et performances de chargement.
- Nombre de sessions et d'utilisateurs.

- Les appels services, temps de réponse et taux d'échec.
- Compteurs de performances des serveurs Windows ou Linux, par exemple le processeur, la mémoire et l'utilisation du réseau.
- Diagnostics d'hébergement Azure.
- Journaux de suivi des diagnostics des applications : pour pouvoir mettre en corrélation les événements de suivi avec les demandes.
- Mesures et événements personnalisés (à implémenter dans le code source)
- Permet d'envoyer des alertes en cas de télémétrie anormales :

 **Azure Application Insights**

 **Augmentation anormale du taux de demandes ayant échoué dans l'application "fabrikamprod"**

Quand cela s'est-il produit ? 12 décembre 2016 11:36 - 11:56 UTC
 Quel est le problème ? **49,5 %** taux de demandes ayant échoué (54/109)
 Taux normal sur 7 jours : **17,7 %**

77,8 % des demandes ayant échoué ont affecté **2 utilisateurs** et possédaient [ces caractéristiques](#) :

Code de réponse :	500 - Erreur de serveur interne
Nom de la demande :	POST Customers/Create
Hôte de l'URL du serveur :	aiconnect2.cloudapp.net
Version de l'application :	AutoGen_29be773e-4ebe-4b96
Instance de rôle :	AIConnect2
Pays :	Royaume-Uni

77,8 % des demandes ayant échoué possédaient [cette exception](#) :

Méthode de l'exception :	FabrikamFiber.DAL.Data.CustomerRepository.Save
Type d'exception :	System.Data.SqlClient.SqlException
Exemple de message :	L'instruction INSERT est en conflit avec la contrainte CHECK « chk_read_only ». Le conflit s'est produit dans la base de données...


77,8 % des demandes ayant échoué avaient des défaillances dans [cette dépendance](#) :


SQL: tcp:fabrikamxyz.database.windows.net,1433 | Fabrikam|SQL

Les demandes ayant échoué caractérisées ci-dessus comportaient des [traces](#) :

- Nouvelle demande reçue
- Début de l'enregistrement du nouveau client

Pour continuer le diagnostic :

 Accédez à [Analytics](#) pour les requêtes puissantes sur votre télémétrie

 Accédez au [portail Application Insights](#) pour les graphiques métriques et la recherche de télémétrie

Cette notification vous a-t-elle été utile ? [Oui](#), [Non](#)

3-3-4-5. Gestion des pannes et résilience

Des mécanismes intégrés à Azure Service Fabric tente d'apporter la plus grande fiabilité possible à notre application :

- Scalabilité automatique suivant des règles de charge
- Load balancer pour répartir la charge
- Redémarrage des services automatique

- Outils de monitoring et de diagnostics pour analyser les problèmes

3.3.4.6. Gestion des transactions

Il s'agit d'une problématique purement logicielle à traiter manuellement avec éventuellement :

- Commit en deux étapes :

voir [https://msdn.microsoft.com/enus/library/aa754091\(v=bts.10\).aspx](https://msdn.microsoft.com/enus/library/aa754091(v=bts.10).aspx)

- Méthode d'annulation : gestion par le développeur d'une méthode Non A pour chaque méthode A.

Ces deux façons de faire sont compliqués à mettre en œuvre et le risque d'erreur est important ... Elles ne sont pas conseillées en général. Après tout si deux services sont impliqués dans une transaction commune, le découpage en « bounded context » de l'approche DDD n'est probablement pas optimal !

En conclusion de ce chapitre, nous pouvons dire qu'Azure Service Fabric propose une solution clé en main pour la conception d'une architecture microservices dans Azure car il offre des réponses aux différentes problématiques de mise en œuvre de microservices.

Grâce à cette offre PAAS, les développeurs ont à leur disposition :

- Un Framework de développement en .Net et des templates de projet pour VS2017
- Une solution CI/CD compatible avec VSTS
- Une offre PAAS pour l'intégration de la solution dans le cloud Azure. Cette offre PAAS garantis :
 - Scalabilité de nos services.
 - Découvrabilité via le Reverse Proxy.
 - Intégration avec API Management pour les appels clients.
 - Monitoring avec Application Insights.
 - Communication avec Azure Service Bus pour les appels asynchrone.

De plus, un des avantages majeurs de cette solution à mon sens est que les développeurs retrouveront leur marque assez rapidement par rapport au développement d'API REST en ASP .Net Core. A contrario une solution comme Azure Container Service implique la montée en compétence sur :

- Docker.
- Un orchestrateur de conteneur Docker.
- La plateforme AKS en elle-même.

4. Mise en œuvre

Nous allons maintenant détailler la mise en œuvre concrète de la solution étudiées ci-dessus.

4.1. Existant

Notre système d'information de démonstration contient une application cœur de métier, utilisée par tous les employés du SI.

Cet applicatif « monolithe » adresse ainsi tous les métiers de l'entreprise :

- Gestion client
- Comptabilité
- Commerce
- RH
- ...

Chaque employé à accès aux divers écrans en fonction de ces droits attribués par rapport à son poste dans l'entreprise.

Un nouveau besoin pour un usage en mobilité conduit notre SI de démonstration à une étude autour de son existant et une évaluation des possibilités d'amélioration.

4.2. Problématiques de l'existant

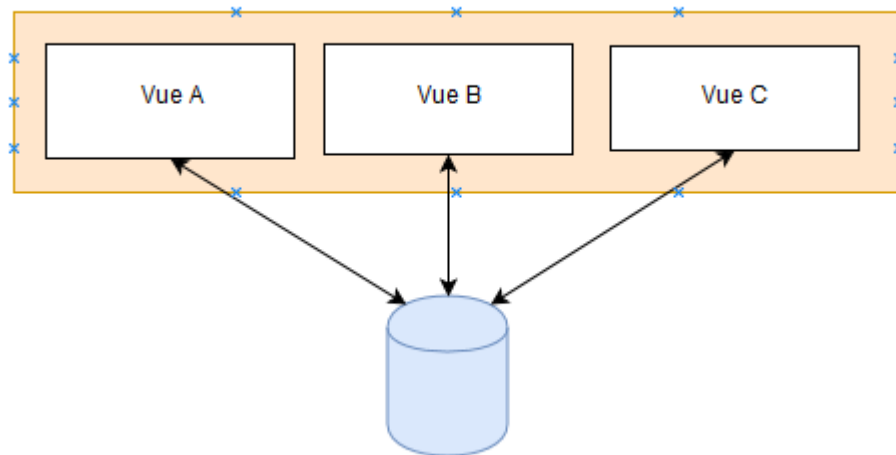
Les problématiques autour de cette application cœur de métier sont nombreuses :

Au niveau du projet et du code .Net, une base de code volumineuse apporte plusieurs problématiques :

- Projet Visual Studio conséquent
- Longueur des builds sur le poste de développement.
- Difficulté à démarrer sur le projet pour un nouveau développeur
- Couplage entre les modules de l'application : code spaghettis

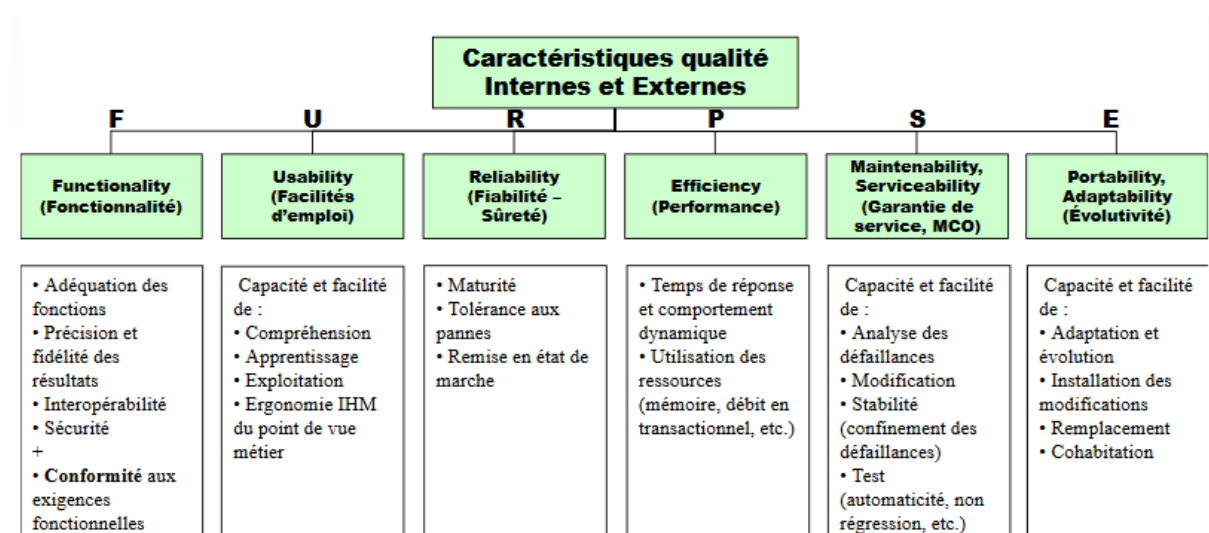
Au niveau de l'application :

- Problèmes de performances.
- En cas de modification, il faut tout retester car les modules sont fortement couplé.
- Cycle en V plutôt que méthode agile lié au coup de déploiement.
- La mise à l'échelle implique de cloner toute l'application sur un nouveau serveur.



L'architecture de notre application

Pour détailler les problèmes de cet existant, nous pouvons utiliser l'approche FURPSE :



- Utilisabilités : L'utilisabilité de l'application est polluée par des lenteurs excessives.
- Fiabilité : En cas de crash, l'utilisateur doit relancer toute l'application
- Performance : Lenteur de l'application, au démarrage et lors de la navigation due aux trop nombreuses dépendances.
- Maintenabilité : En cas d'erreur, difficulté d'analyse.
- Évolutivité : Les évolutions sont coûteuses à mettre en place car elles impliquent beaucoup d'analyse d'impact et de tests de la part des développeurs.

4.3. Conceptions

Le but de ce chapitre est de présenter comment bâtir une architecture microservices à partir de notre existant. Ce que nous voulons obtenir c'est :

Une application de présentation, la plus légère possible et qui ne contient aucune logique fonctionnelle :

- Projet « client lourd » : Demo.Techs.Win
- Projet « client léger » : Demo.Techs.Web

Une couche de service déployé dans Azure Service Fabric :

- Projet Demo.Techs.Microservices

Celui-ci contient des services regroupés par domaine fonctionnels :

- Clients
- Comptabilites
- Documents
- ...

Certains services n'ayant pas besoin d'être forcément scalable (car peu utilisé) je ne les ai pas intégrés à Azure Service Fabric. Ce projet est donc déployé dans une Web App :

- Projet Demo.Techs.API

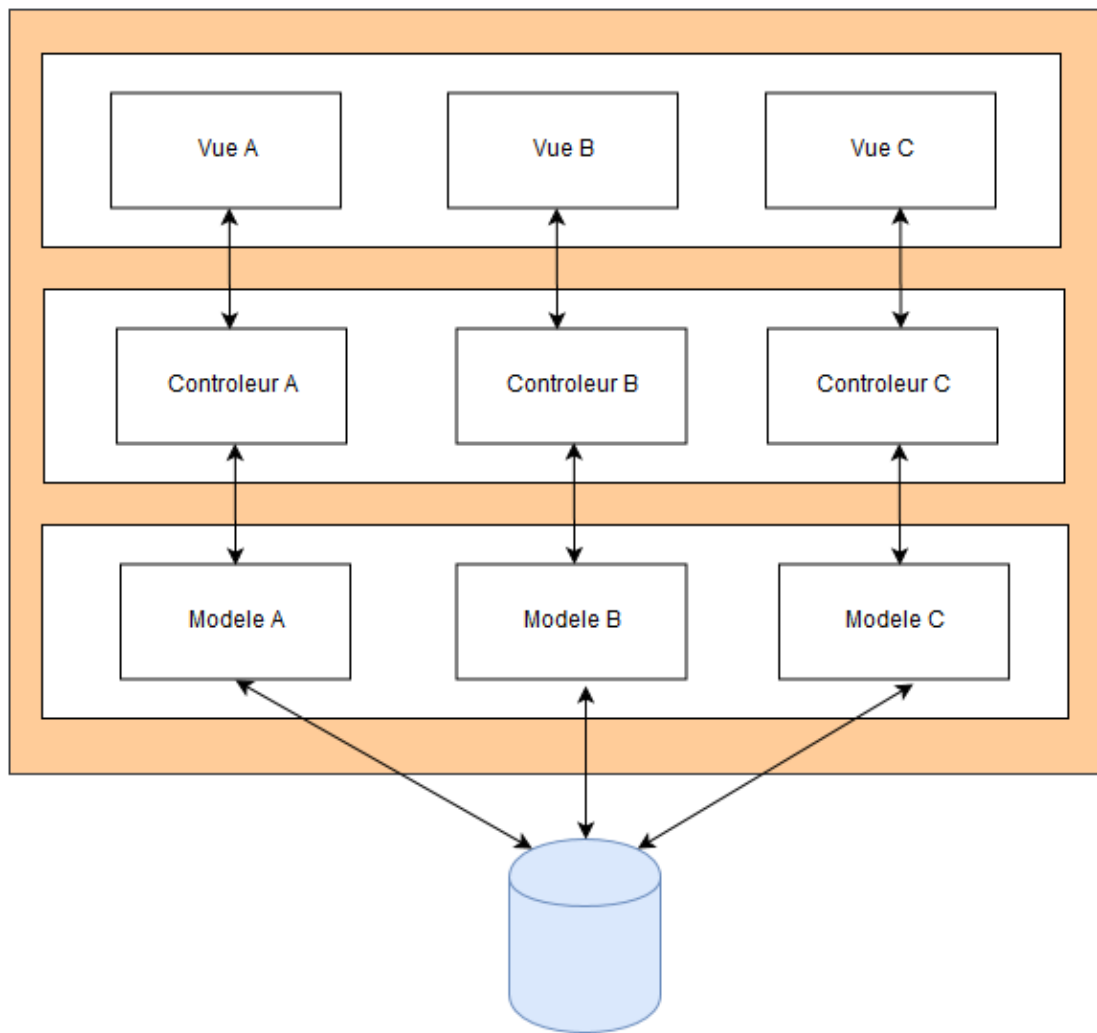
Une fois la logique fonctionnelle extraite de notre application, il sera bien plus facile :

- De faire évoluer nos services.
- D'ajouter une nouvelle application cliente.
- De tester et déployer nos services.
- De les rendre résilient aux pannes et adaptable à la charge.

Nous allons présenter différentes possibilités pour mettre en œuvre notre architectures microservices en gardant à l'esprit nos besoins. Nous souhaitons :

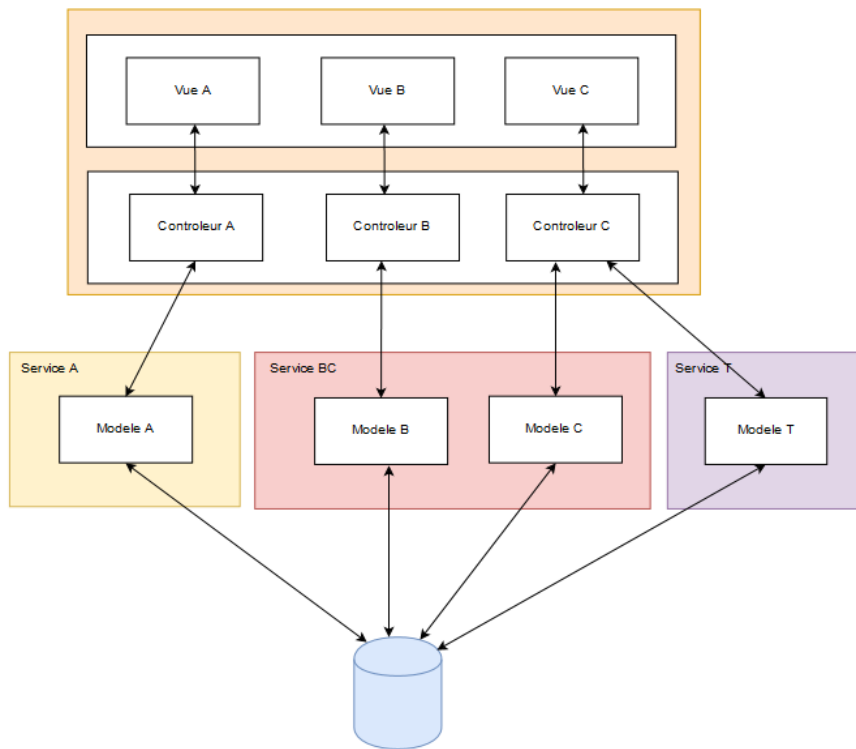
- Des services au contexte délimité.
- Un couplage faible entre composants.
- Testables facilement et automatiquement.
- Déployable automatiquement.

Proposition 1 : Un monolithe organisé :

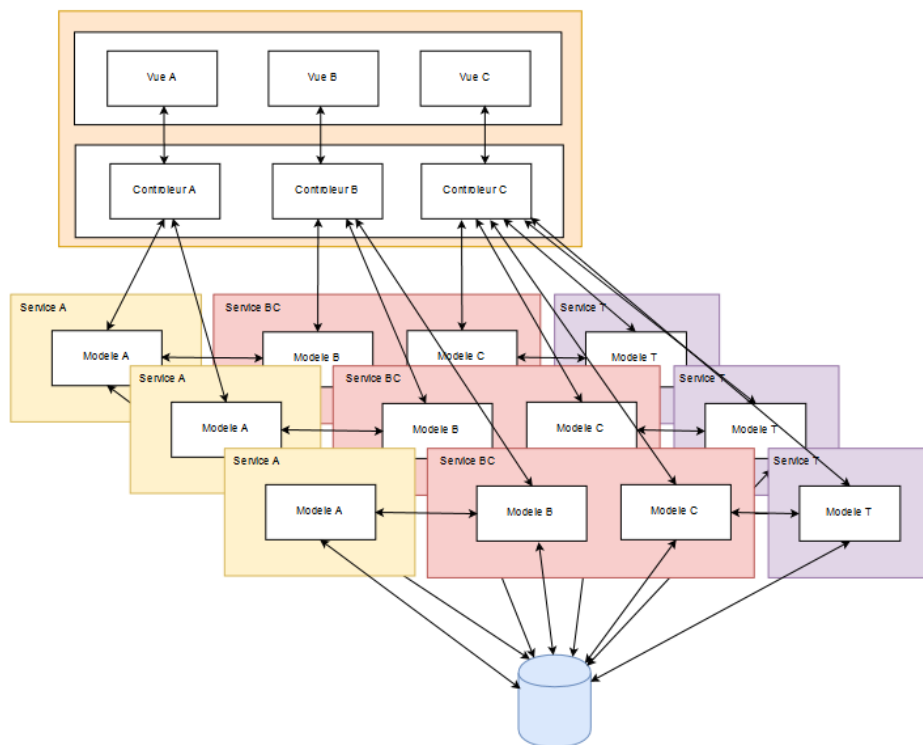


Ce découpage ne convient pas à nos besoins pour des raisons évidente.

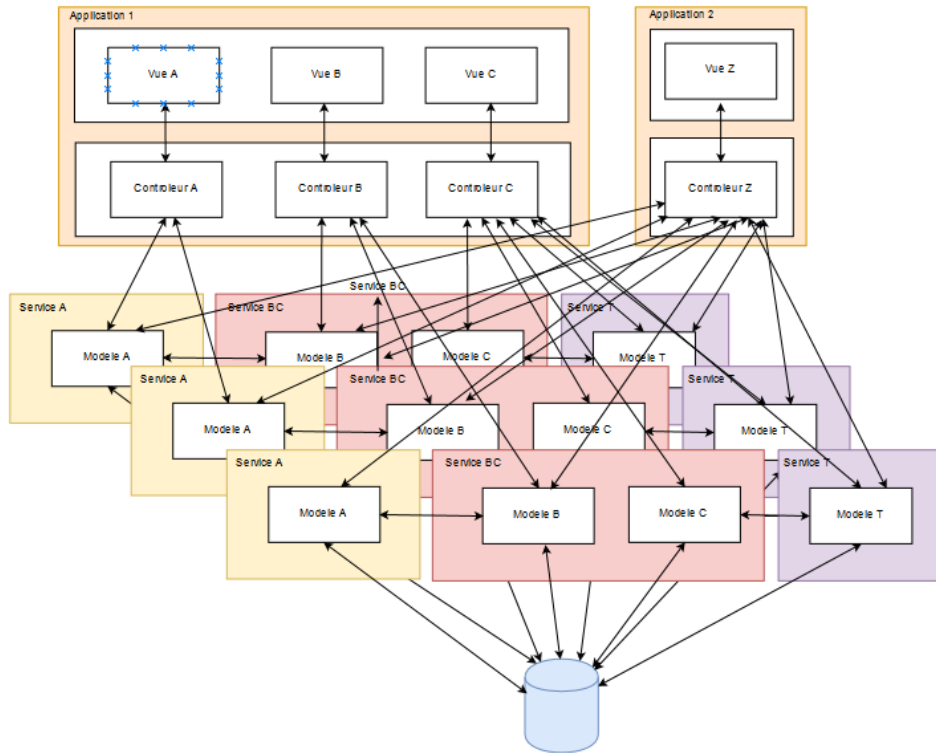
Proposition 2 : Un monolithe et des services :



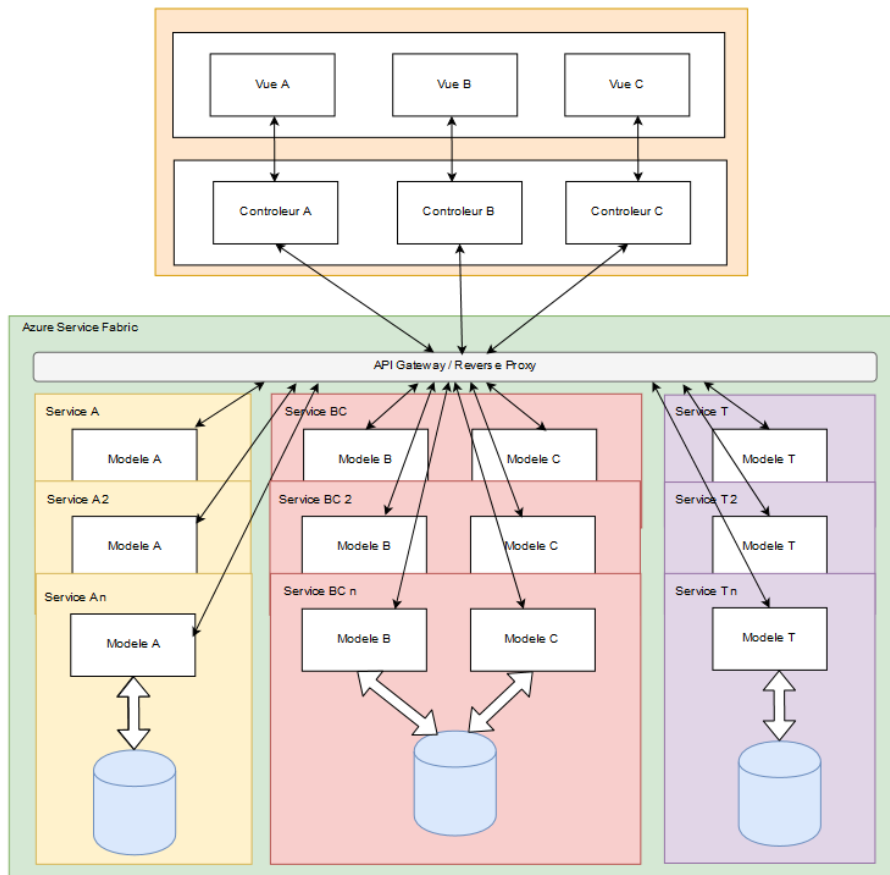
Proposition 3 : Un monolithe distribué. Les services sont dupliqués manuellement pour répondre à la charge :



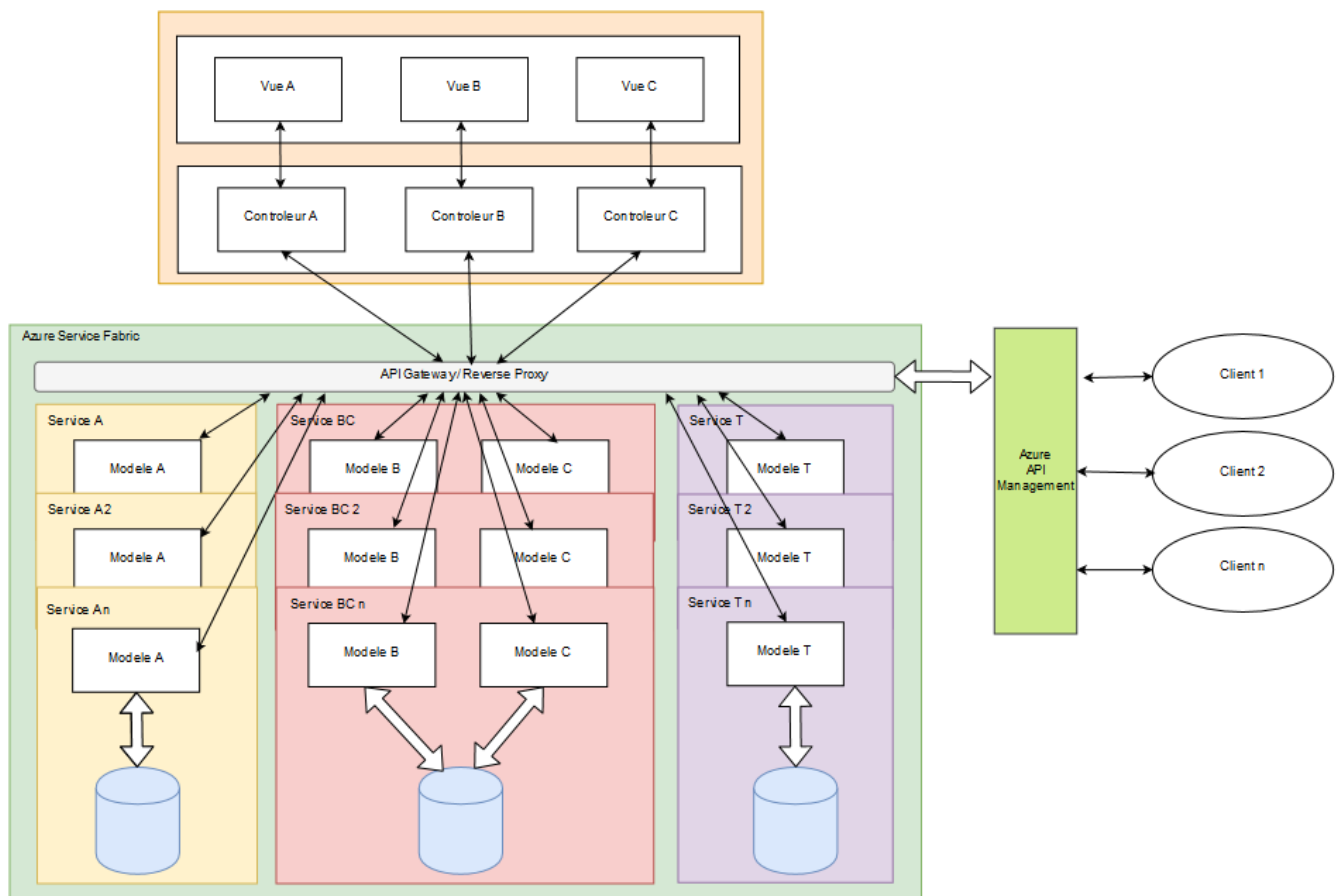
Le risque de cette dernière proposition est de vite se retrouver à gérer :



Proposition 5 : Une architecture microservice avec Azure Service Fabric :



Proposition finale : Si le besoin d'ouvrir notre SI à des clients externe est présent nous pouvons facilement le mettre en œuvre :



4.4. Implémentations

4.4.1. Architecture d'un service

Pour le développement des différents services, une bonne pratique consiste à implémenter le pattern Repository. Notre application se décomposera en 3 couches :

- Une couche Service qui expose les points d'entrée. Elle peut être également responsable de la gestion des exceptions, de l'authentification et des logs.
- Une couche Business qui contient la logique fonctionnelle. Elle organise les context de connexion à la base (UnitOfWork) et appelle les repository pour effectuer des opérations sur une base.
- Une couche Repository qui accède ou modifie les données de la base.

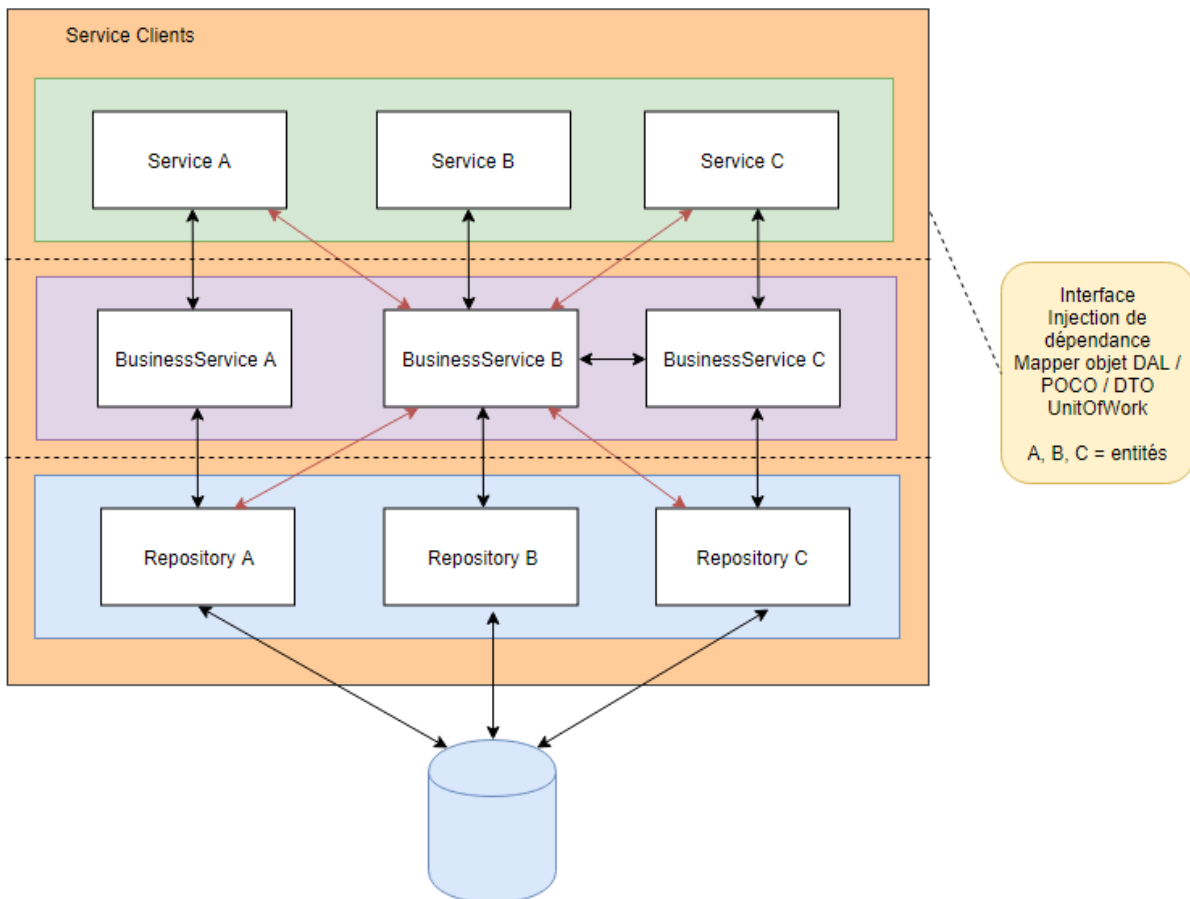


Schéma d'architecture d'un service. Les fleches en rouge représente des appels non autorisés.

Chaque couche est séparée par des interfaces pour la mise en œuvre de l'injection de dépendance dans nos projets. A noter que Asp .Net Core supporte nativement cette bonne pratique via la classe ServiceCollection. Voir : <https://docs.microsoft.com/en-us/aspnet/core/fundamentals/dependency-injection?view=aspnetcore-2.1>

Au niveau des framework et outils tiers nous pouvons utiliser :

- AutoMapper : permet de mapper facilement un objet Entity Framework avec un POCO ou DTO pour le transport des informations par exemple.
- Swagger pour la présentation de nos API. Une valeur sûre ! Voir <https://docs.microsoft.com/en-us/aspnet/core/tutorials/web-api-help-pages-using-swagger?tabs=visual-studio&view=aspnetcore-2.1>
- Newtonsoft JSON .Net : pour la gestion de flux xml.
- Postman pour les tests locaux de communication avec notre API.
- Nuget pour le partage des interfaces.

4.4.2. Création et configuration de Service Fabric

Nous allons maintenant détaillé la création de notre Service Fabric.

1ere étape : paramètres de bases :

Accueil > Nouveau > Cluster Service Fabric > Créer un cluster Service Fabric > Bases

Créer un cluster Service Fabric

Bases

Paramètres de cluster de base

- 1 Bases**
Terminé
- 2 Configuration de cluster**
Définir la configuration du clus...
- 3 Sécurité**
Configurer les paramètres de s...
- 4 Résumé**
Vérifier, afficher le modèle, créer

* Nom de cluster ⓘ
orangems ✓
.westcentralus.cloudapp.azure.com

* Système d'exploitation
WindowsServer 2016-Datacenter ▾

Informations d'identification de machine virtuelle par défaut

* Nom d'utilisateur ⓘ
Thomas

* Mot de passe ⓘ
.....

* Confirmer le mot de passe
.....

* Abonnement
Visual Studio Enterprise ▾

* Groupe de ressources
 Créer nouveau Utiliser existant

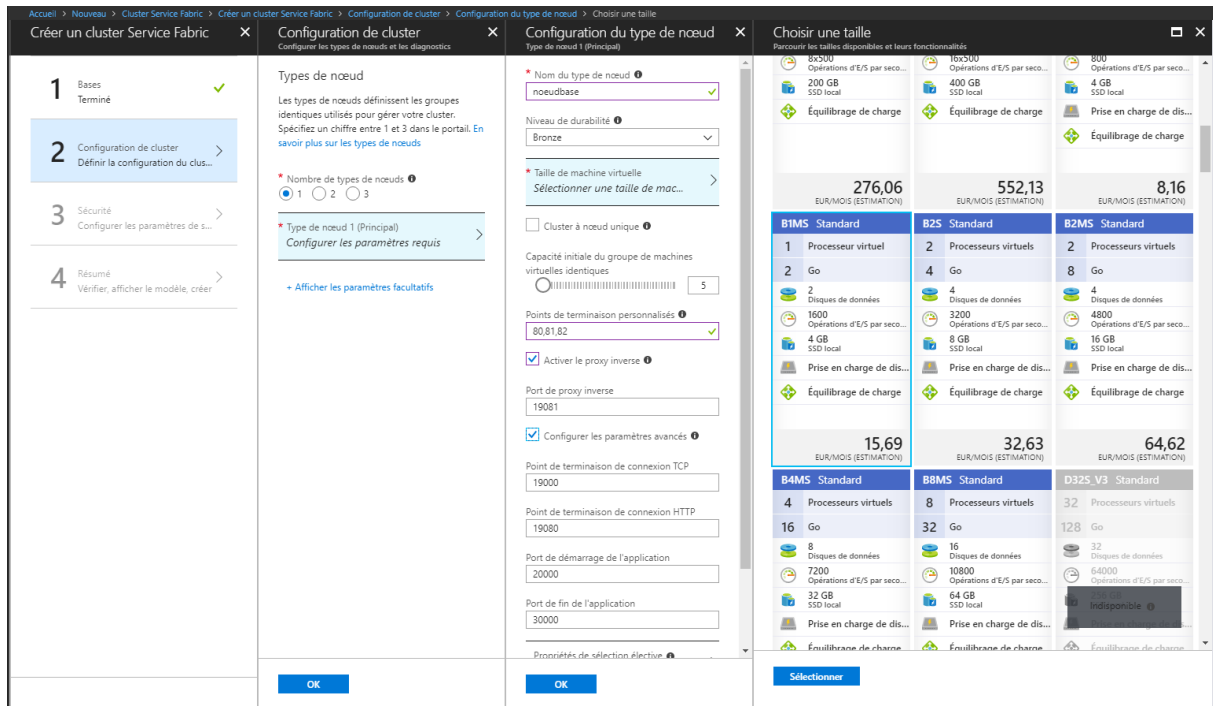
ressourcesorangems ✓

* Emplacement
Ouest-Centre des États-Unis ▾

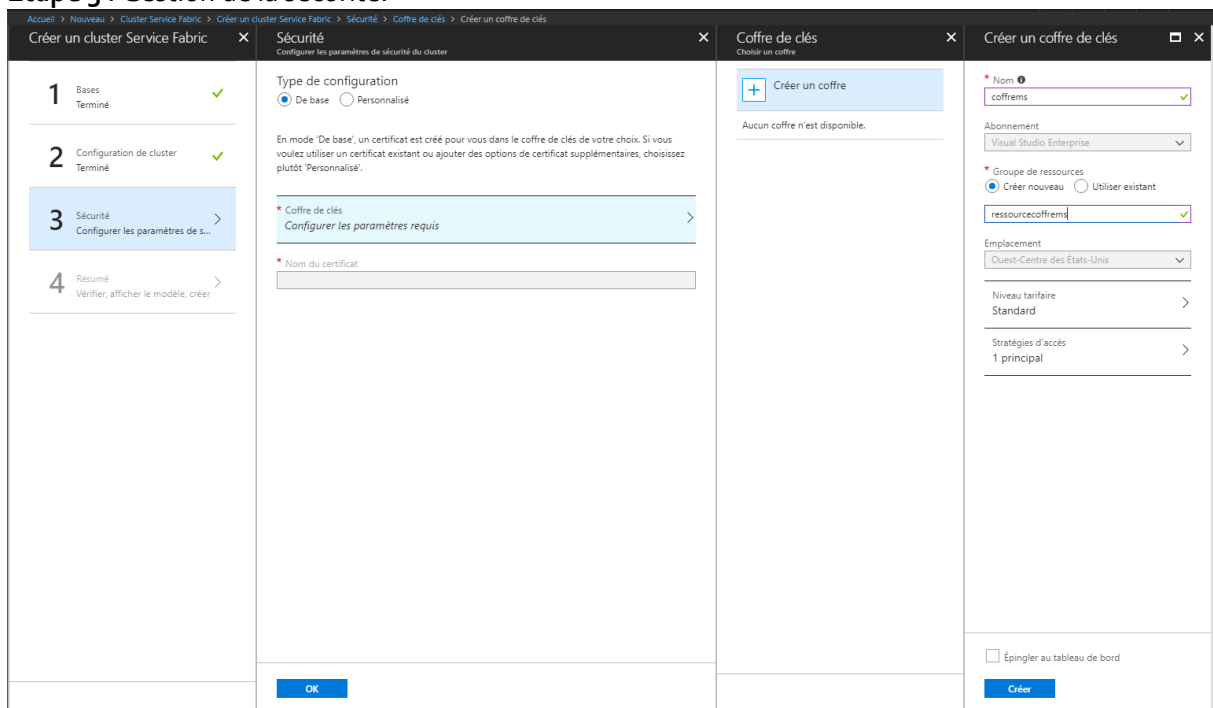
OK

2eme étape : configuration du cluster. On doit ici :

- Choisir le nombre de nœud de notre cluster
- Pour chaque nœud on configure la machine hôte et les paramètres réseaux. Attention à la configuration des ports qui seront accessibles (80,81,82 ici) et à bien activer le reverse proxy !



Etape 3 : Gestion de la sécurité.



Il faut créer un coffre de sécurité qui contiendra le certificat d'accès à notre cluster, modifier les stratégies d'accès puis valider.

Accueil > Nouveau > Cluster Service Fabric > Créer un cluster Service Fabric > Sécurité > Stratégies d'accès

Créer un cluster Service Fabric x

- 1 Bases Terminé ✓
- 2 Configuration de cluster Terminé ✓
- 3 Sécurité Configurer les paramètres de s... >
- 4 Résumé Vérifier, afficher le modèle, créer >

Sécurité Configurer les paramètres de sécurité du cluster x

Type de configuration
 De base Personnalisé

En mode 'De base', un certificat est créé pour vous dans le coffre de clés de votre choix. Si vous voulez utiliser un certificat existant ou ajouter des options de certificat supplémentaires, choisissez plutôt 'Personnalisé'.

Coffre de clés coffres ! >

! Le coffre de clés que vous avez sélectionné n'est pas activé pour le déploiement. Pour l'activer, vous devez en modifier les stratégies d'accès : sélectionnez l'option 'Activer l'accès aux machines virtuelles Azure pour le déploiement' qui se trouve dans la section 'Stratégies d'accès avancées' et cliquez sur 'Enregistrer'. Si vous n'avez pas l'autorisation de modifier le coffre de clés sélectionné, vous devez contacter son propriétaire pour appliquer la modification ou choisir un autre coffre de clés.

[Modifier les stratégies d'accès de coffres](#)

* Nom du certificat

! Réglez les erreurs sur cette page avant de continuer.

OK

Stratégies d'accès x

Enregistrer Ignorer Actualiser

[Cliquez ici pour masquer les stratégies d'accès avancées](#)

- Activer l'accès aux machines virtuelles Azure pour le déploiement
- Activer l'accès à Azure Resource Manager pour le déploiement de modèles
- Activer l'accès à Azure Disk Encryption pour chiffrer des volumes

+ Ajouter nouveau ...

10491c74-816e-4413-a2f4-... UTILISATEUR

Créer un cluster Service Fabric x

- 1 Bases Terminé ✓
- 2 Configuration de cluster Terminé ✓
- 3 Sécurité Configurer les paramètres de s... >
- 4 Résumé Vérifier, afficher le modèle, créer >

Sécurité Configurer les paramètres de sécurité du cluster x

Type de configuration
 De base Personnalisé


En mode 'De base', un certificat est créé pour vous dans le coffre de clés de votre choix. Si vous voulez utiliser un certificat existant ou ajouter des options de certificat supplémentaires, choisissez plutôt 'Personnalisé'.



* Coffre de clés coffres >


* Nom du certificat
 ✓

Validation... O

Téléchargez le certificat créé, il nous servira ultérieurement.

 **c73236279f7e4ffda33c6bc99e4075b4** ✦ □ ✕
Version du secret

 Enregistrer  Ignorer


 La durée de vie de ce secret est gérée par Azure Key Vault, dont elle ne peut pas être directement modifiée ici. Cliquez ici pour afficher et modifier son certificat correspondant. [→](#)

Propriétés


Créé 09/04/2018 à 22:34:16

Mis à jour 09/04/2018 à 22:34:16


Identificateur de secret





Paramètres

Définir la date d'activation ? 


Date d'activation




(UTC+02:00) --- Fuseau horaire actuel --- 


Définir la date d'expiration ? 

Date d'expiration



(UTC+02:00) --- Fuseau horaire actuel --- 

Activé ?

Balises 

Balises 0

Secret

Type de contenu (facultatif)

[Télécharger sous la forme d'un certificat](#)

Téléchargez le template généré avant de valider : ce template pourra être utilisé pour recréer notre cluster :

Modèle

Télécharger Ajouter à la bibliothèque Déployer

Automatisez le déploiement de ressources avec des modèles Azure Resource Manager en une seule opération coordonnée. Définissez des ressources et des paramètres d'entrée configurables, et effectuez les déploiements.

Modèle Paramètres CLI PowerShell .NET Ruby

```

1 {
2   "$schema": "http://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json",
3   "contentVersion": "1.0.0.0",
4   "parameters": {
5     "clusterLocation": {
6       "type": "string",
7       "metadata": {
8         "description": "Location of the Cluster"
9       }
10    },
11    "clusterName": {
12      "type": "string",
13      "defaultValue": "Cluster",
14      "metadata": {
15        "description": "Name of your cluster - Between 3 and 23 characters. Letters and numbers only"
16      }
17    },
18    "ntApplicationStartPort": {
19      "type": "int",
20      "defaultValue": 20000
21    },
22    "ntApplicationEndPort": {
23      "type": "int",
24      "defaultValue": 30000
25    },
26    "ntOperationalStartPort": {
27      "type": "int",
28      "defaultValue": 49152
29    },
30    "ntOperationalEndPort": {
31      "type": "int",
32      "defaultValue": 65534
33    },

```

template.zip

Et il n'y a plus qu'à valider le déploiement de notre plateforme :

Créer un cluster Service Fabric Résumé

1 Bases Terminé ✓

2 Configuration de cluster Terminé ✓

3 Sécurité Terminé ✓

4 Résumé Vérifier, afficher le modèle, créer >

Validation réussie

Bases

Abonnement	Visual Studio Enterprise
Groupe de ressources	ressourcesorangems
Emplacement	Ouest-Centre des États-Unis

Paramètres

Nom de cluster	orangems
Nom d'utilisateur	Thomas
Nombre de types de nœuds	1
Créer le stockage de journal d'a...	Activé

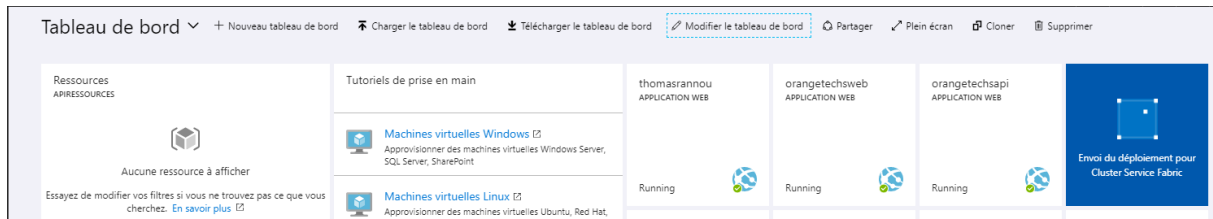
Types de nœud

Type de nœud 1 (Principal)	noeudbase (5xStandard_B1ms)
Taille de machine virtuelle	Standard_B1ms
Points de terminaison person...	80,81,82

Vous avez besoin de votre nouveau certificat pour vous connecter à votre cluster. Vous pouvez afficher et télécharger votre certificat en cliquant sur [ce lien](#).

Créer Télécharger le modèle et les paramètres

L'installation est en cours :

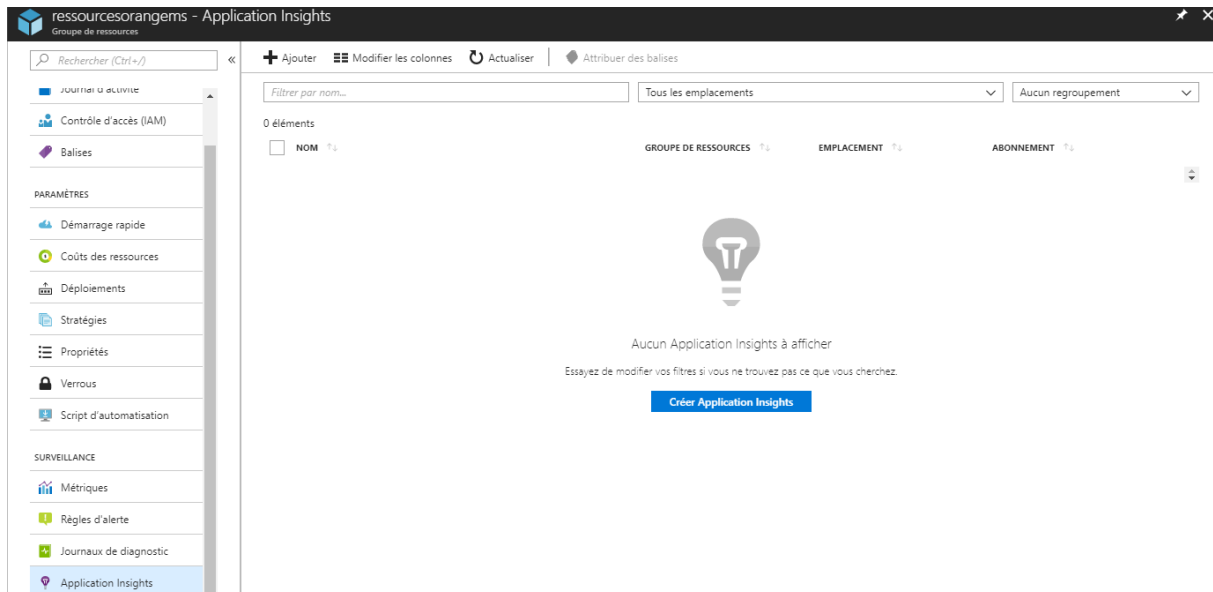


Une fois toutes les étapes validées, Azure crée notre Service Fabric. L'opération dure un certain temps, 10 minutes environ.

Il faut maintenant installer le certificat précédemment téléchargé pour pouvoir accéder à l'interface d'administration Service Fabric. Il faut installer le certificat dans le magasin de certificat Windows et dans le navigateur utilisé.

Il nous reste maintenant à déployer Application Insights pour notre cluster.

Pour se faire, positionnez-vous sur votre ressource Service Fabric et cliquez sur 'Créer Application Insights' :



Application Insights

Analysez les performances et l'utilisation d'une ...



Nom ⓘ

Alorangems ✓

* Type d'application ⓘ

Application Web ASP.NET ▼

* Abonnement

Visual Studio Enterprise ▼

* Groupe de ressources ⓘ

Créer nouveau Utiliser existant

ressourcesaiorangems ✓

* Emplacement

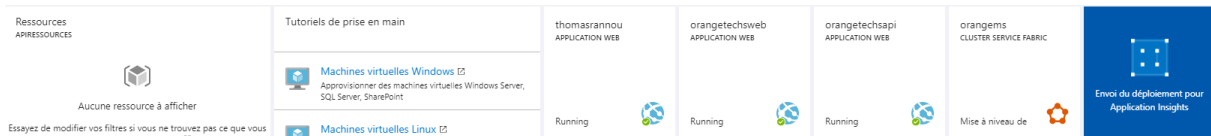
Est des États-Unis ▼

Épingler au tableau de bord

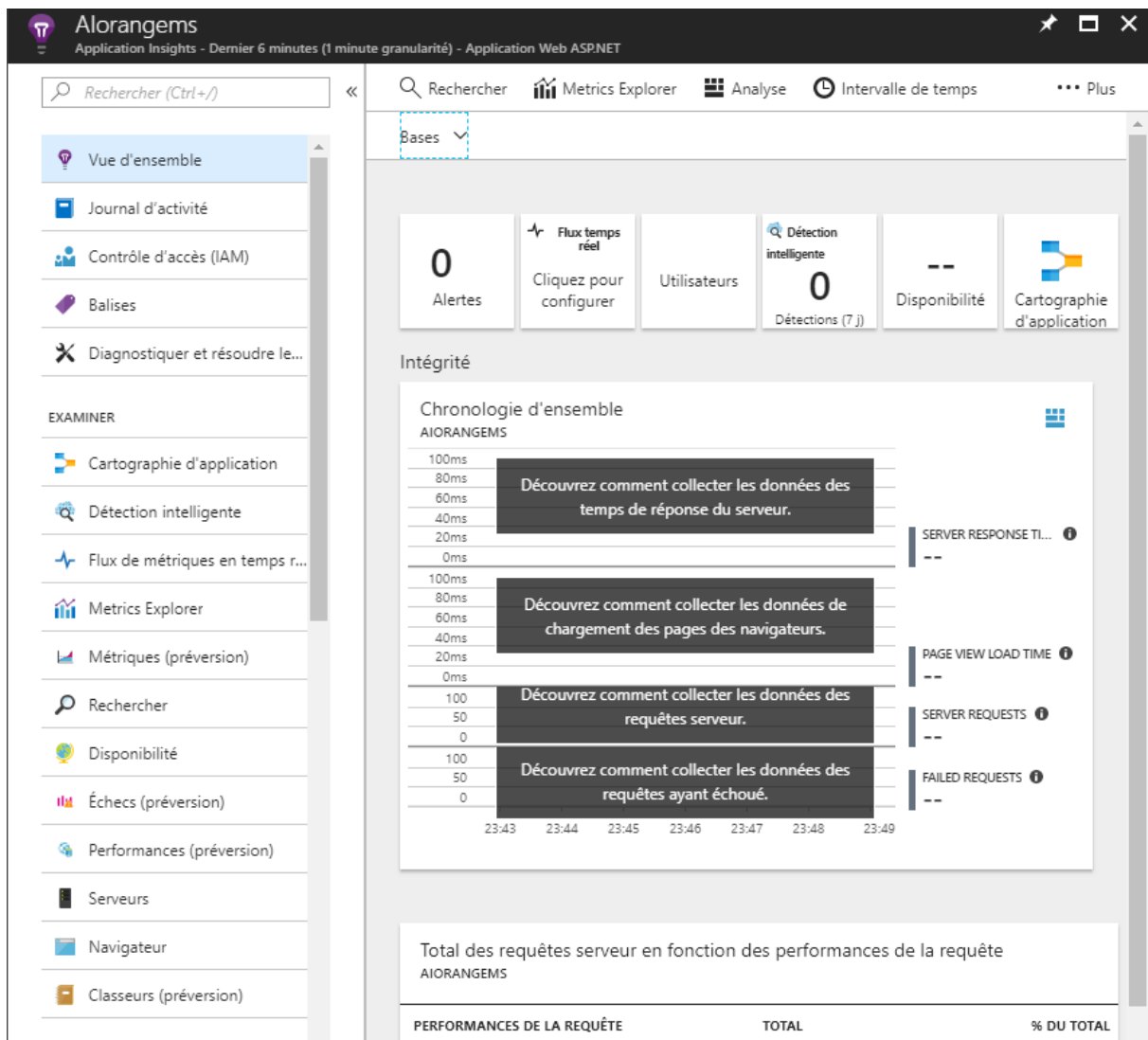
Créer

[Options d'automatisation](#)

Le déploiement d'Insights est en cours :



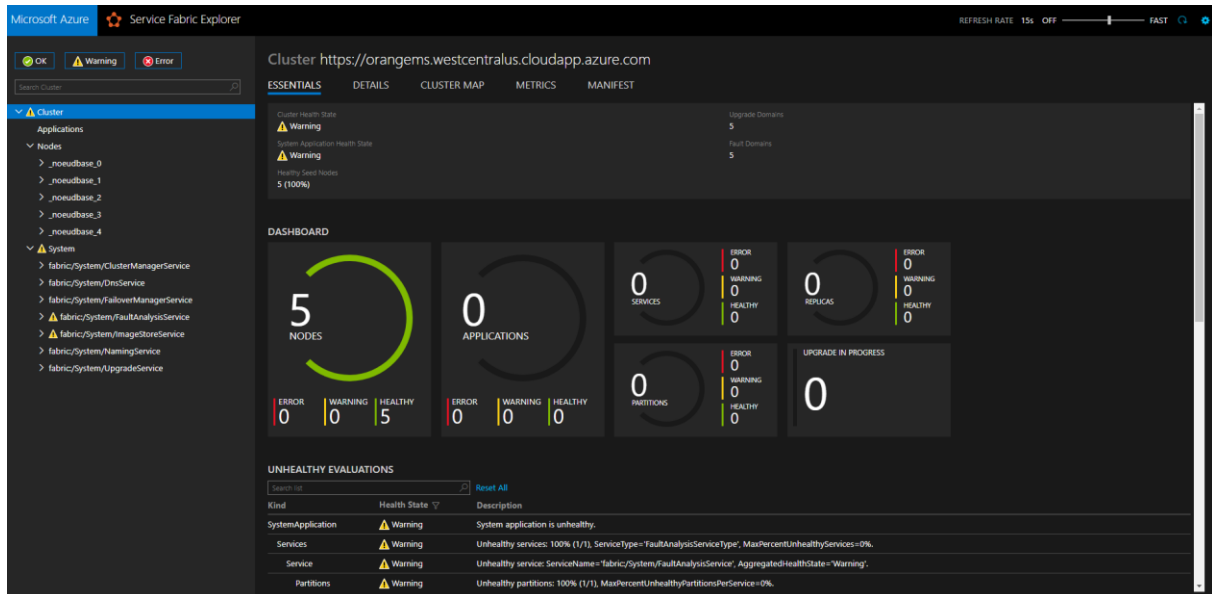
Nous avons maintenant accès au dashboard de configuration et de monitoring d'Insights pour notre cluster Service Fabric et nos services.



Voici l'interface de projet : Service Fabric Explorer. Cet outils permet de visualiser rapidement l'état de santé de notre cluster et des instances des applications.

L'url de cet outils est : <https://nom-du-cluster.localisation-du-cluster.cloudapp.azure.com:19080/Explorer>

Exemple : <https://demoms.westcentralus.cloudapp.azure.com:19080/Explorer/>

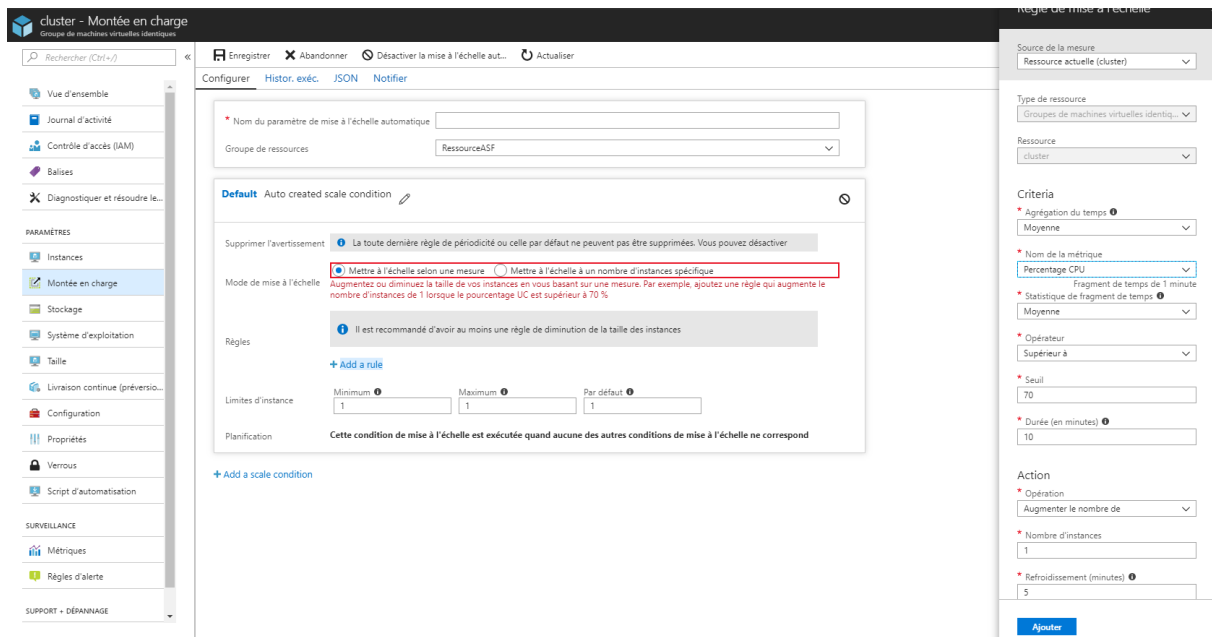


Attention à l'ouverture des ports sur le cluster et sur chaque nœud pour l'accessibilité externe à Service Fabric. Pour accéder au service, il faudra ajouter des règles de routage dans l'équilibreur de charge.

The screenshot shows the Azure portal interface for configuring load balancer rules. The page title is "LB-orangems-noeudbase - Règles d'équilibrage de charge". The table below lists the configured rules:

NOM	RÈGLE D'ÉQUILIBRAGE DE CHARGE	POOL PRINCIPAL	SONDE D'INTÉGRITÉ
AppPortLBRule1	AppPortLBRule1 (TCP/80)	LoadBalancerBEAddressPool	AppPortProbe1
AppPortLBRule2	AppPortLBRule2 (TCP/81)	LoadBalancerBEAddressPool	AppPortProbe2
AppPortLBRule3	AppPortLBRule3 (TCP/82)	LoadBalancerBEAddressPool	AppPortProbe3
LBHttpRule	LBHttpRule (TCP/19080)	LoadBalancerBEAddressPool	FabricHttpGatewayProbe
LBRule	LBRule (TCP/19000)	LoadBalancerBEAddressPool	FabricGatewayProbe

L'écran « Montée en charge » permet de renseigner des règles pour adapter le nombre d'instances de nos services disponibles suivant la charge :



Par exemple, nous pouvons définir des règles par rapport :

- Au trafic réseau
- Un seuil d'utilisation CPU ou mémoire
- Au nombre d'appels à nos API ...

Nos cluster Service Fabric est maintenant opérationnel dans Azure et grâce à l'Explorer nous pouvons valider son bon fonctionnement même si pour le moment aucune application n'est déployé.

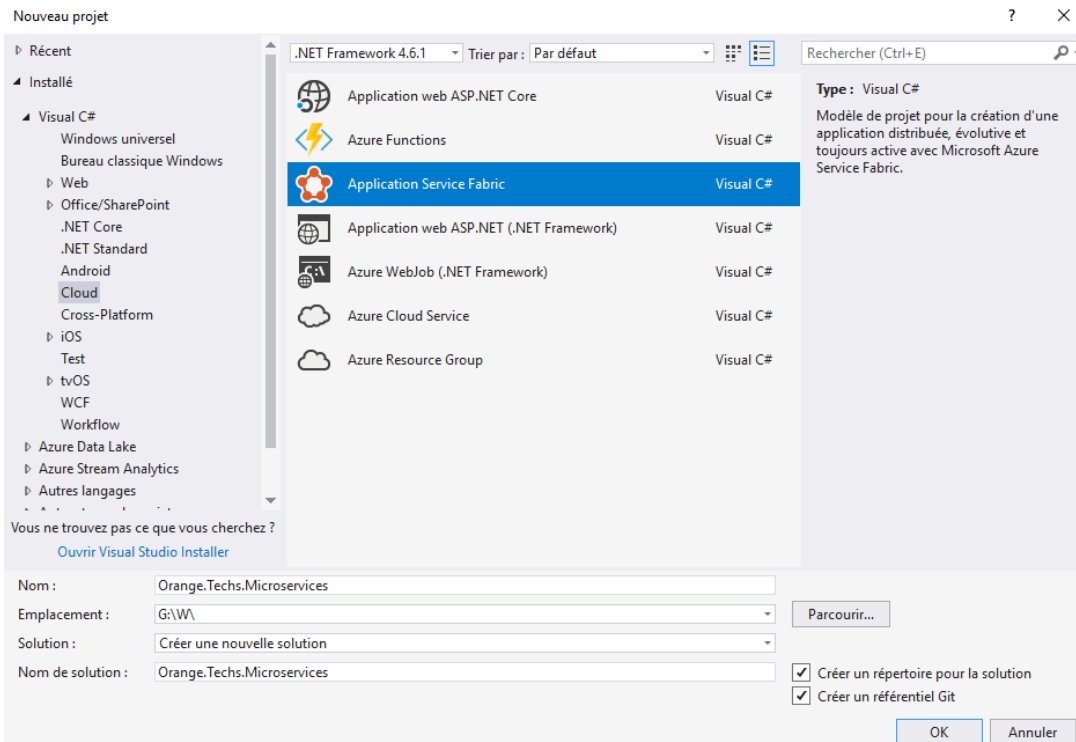
4.4.3. Projet Visual Studio Service Fabric

Nous allons maintenant détailler la creation de notre projet Service Fabric dans Visual Studio 2017.

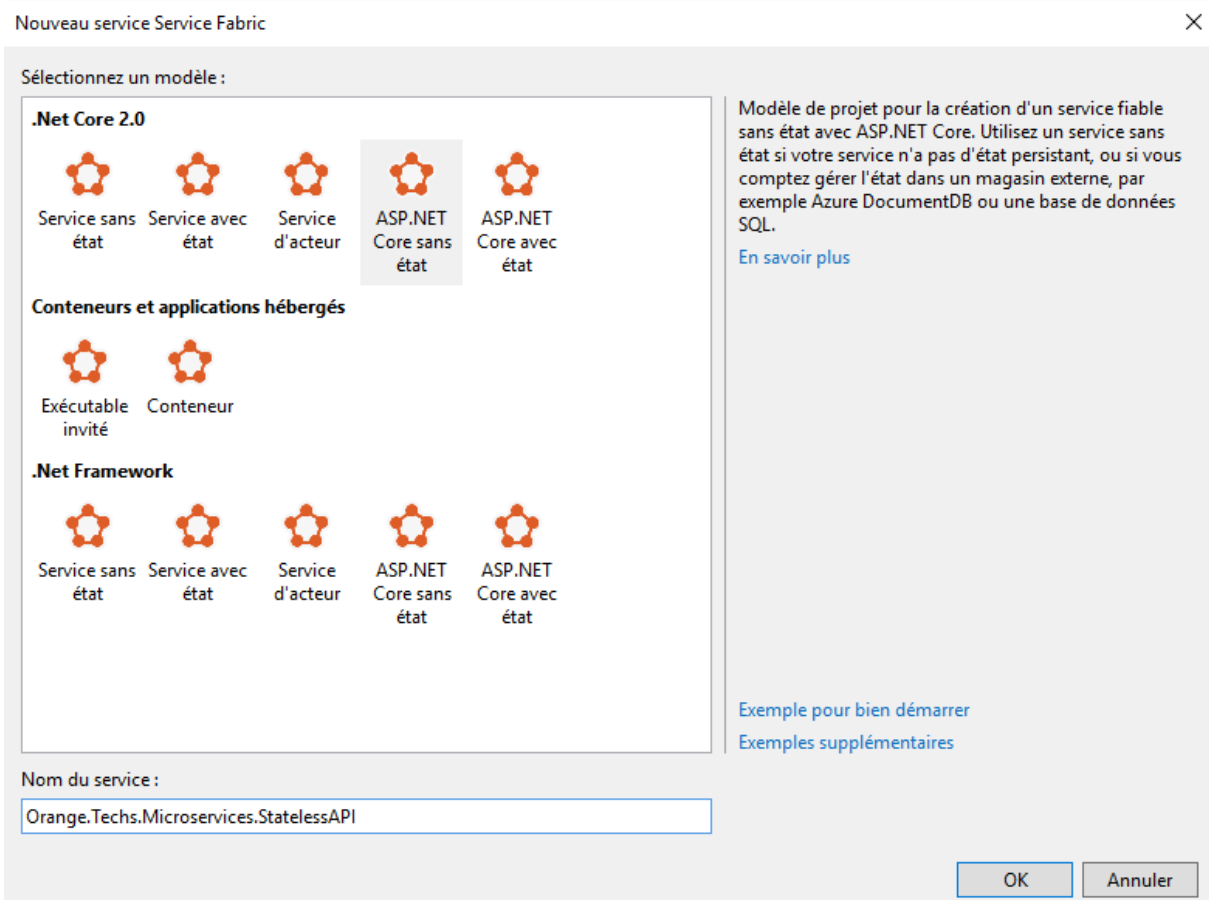
4.4.3.1. Création du projet

Tout d'abord il faut configurer le poste du développeur pour utiliser les templates de projets ASF. Pour télécharger et installer le SDK rendez vous ici : <https://docs.microsoft.com/fr-fr/azure/service-fabric/service-fabric-get-started>

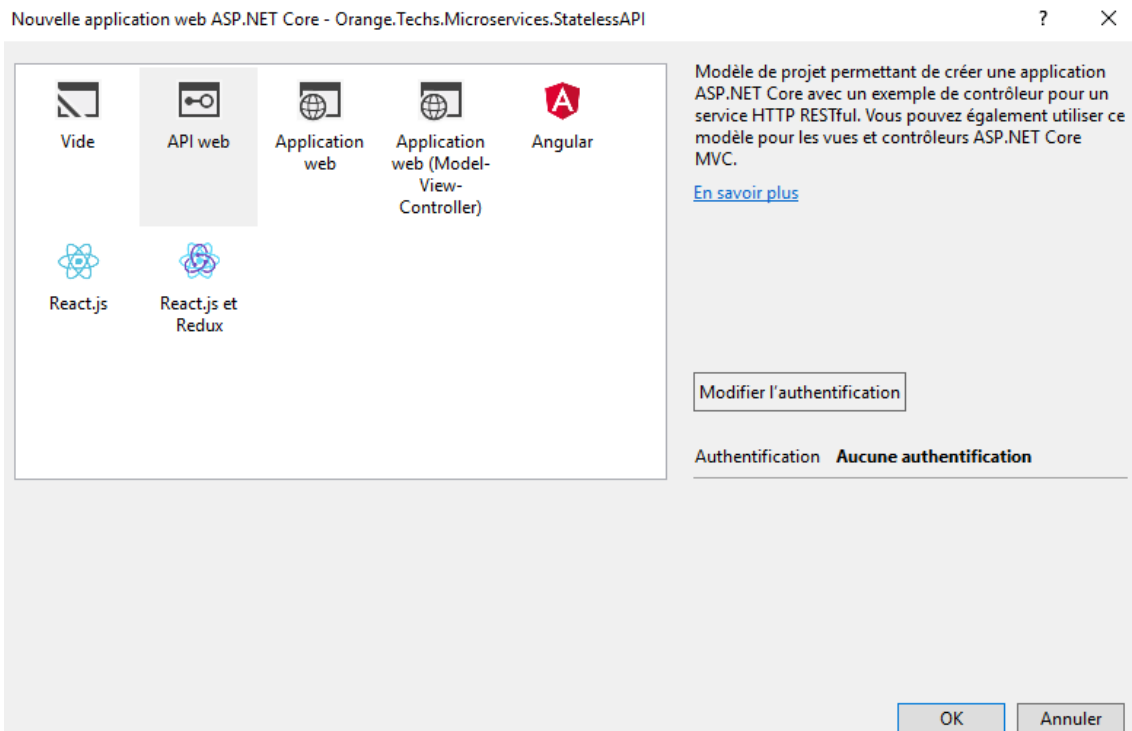
Nous pouvons maintenant créé noter projet Cloud de type Application Service Fabric :



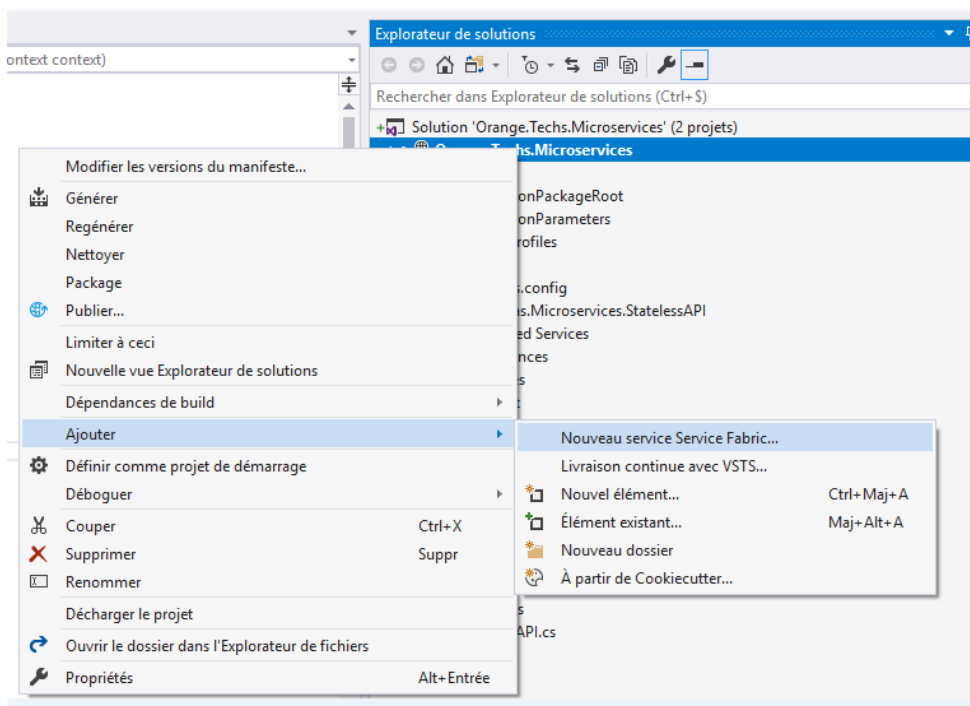
Nous devons ajouter un premier service à notre projet :



Je choisis le type de projet ASP .Net Core que je veux implémenter comme pour un projet .Net Core classique :



Si je veux ajouter un nouveau projet à mon application il faut faire clic droit sur notre projet principal et Ajouter > Nouveau Service Service Fabric pour que ce nouveau projet soit lié à notre environnement ASF.

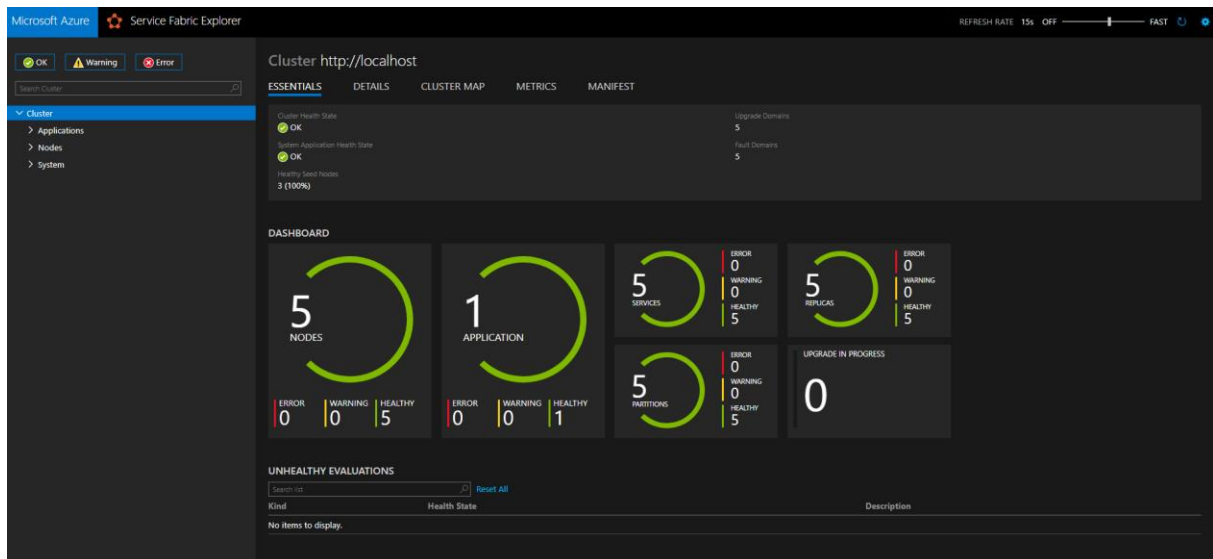
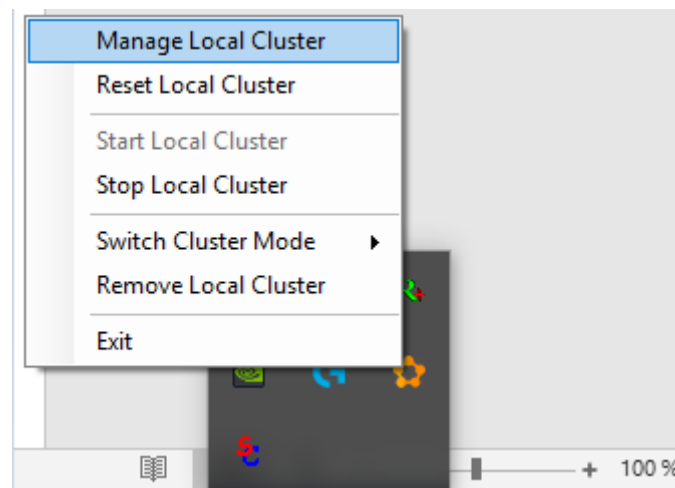


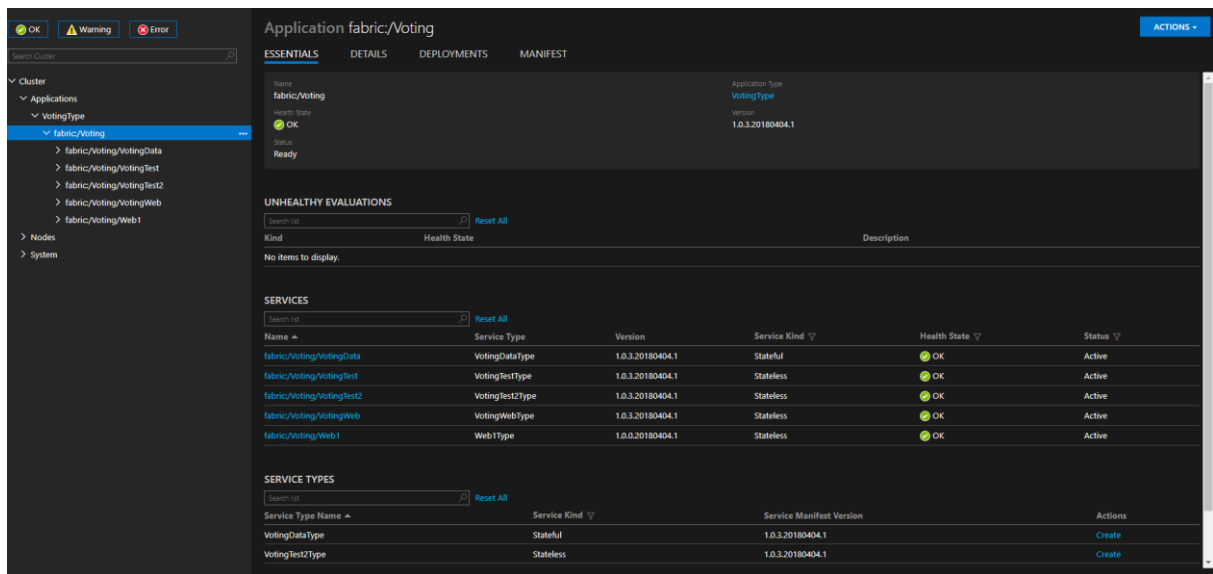
On voit ici que la différence principale entre un service ASP .Net Core standard et un service ASP .Net Core dans Service Fabric c'est le projet par défaut créé par Visual Studio. Celui-ci contient des fichiers

de configurations pour le build et déploiement de nos services dans un cluster Service Fabric. Par conséquent, vous retrouvez facilement vos habitudes de développement au niveau des services ASP .Net Core !

4.4.3.2. Tests en local

En local il est possible de tester son projet Azure Service Fabric comme si il était déployé dans le Cloud puisque le déploiement de notre application se fait dans un cluster Service Fabric local. On a ensuite accès à l'écran d'administration et nous pouvons debugger notre application comme n'importe quel projet .Net.

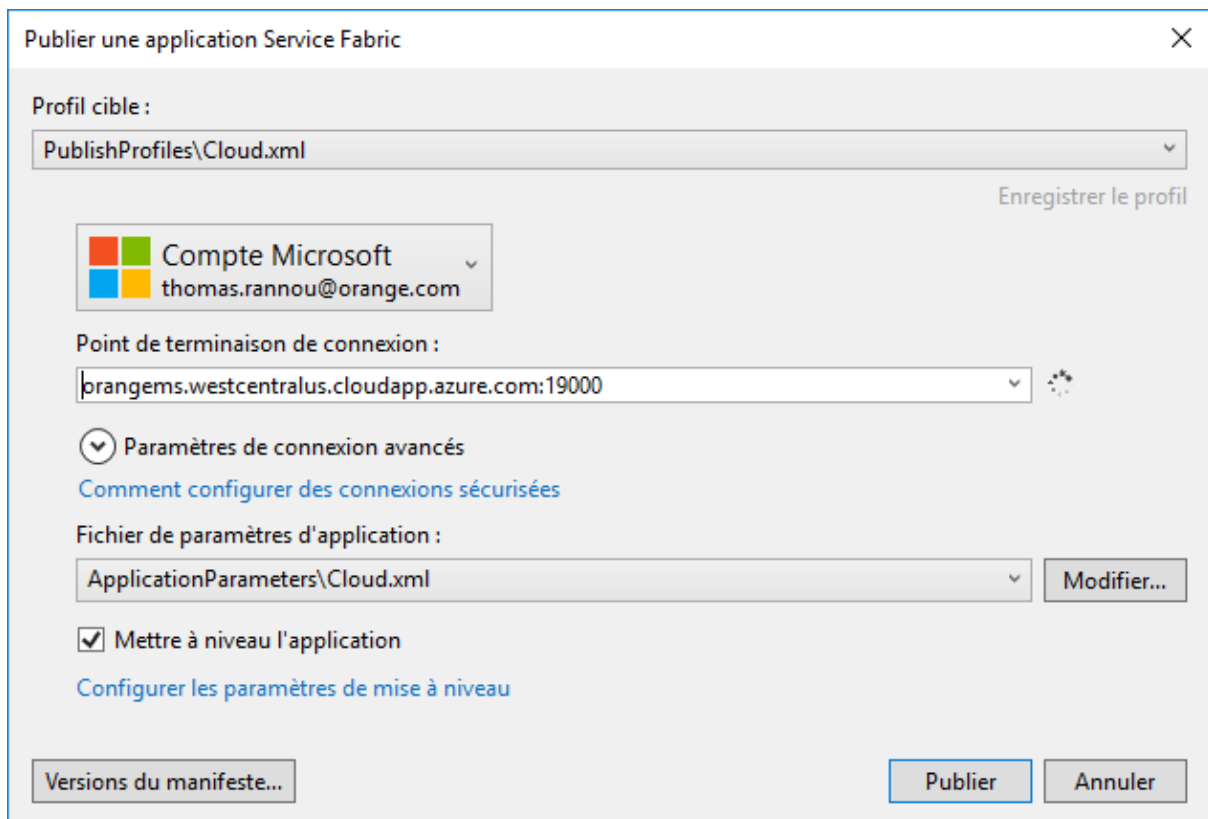




Le monitoring de nos services en local

4.4.3.3. Premiers déploiements

Pour effectuer nos premiers tests de déploiement, on peut utiliser la publication Visual Studio qui s'interface très bien avec Azure. Attention pour les publication de « mise à jour » pensez à cocher « Mettre à niveau l'application » comme ci-dessous et à modifier les numéros de versions du manifeste.



Pour le déploiement vers notre cluster, il faudra configurer le fichier cloud.xml :

```

23 <ClusterConnectionParameters
24   ConnectionEndpoint="orangems.westcentralus.cloudapp.azure.com:19000"
25   X509Credential="true"
26   ServerCertThumbprint="8C4AC5ED97DA2B1E66DA19335343F85AAAAAAAAAAAA"
27   FindType="FindByThumbprint"
28   FindValue="8C4AC5ED97DA2B1E66DA19335343F85AAAAAAAAAAAA"
29   StoreLocation="CurrentUser"
30   StoreName="My" />
31 <ApplicationParameterFile Path="..\ApplicationParameters\Cloud.xml" />
32 <UpgradeDeployment Mode="Monitored" Enabled="true">
33   <Parameters FailureAction="Rollback" Force="True" />
34 </UpgradeDeployment>
35 </PublishProfile>

```

- ConnectionEndpoint : L'URL de notre cluster sur le port 19000
- ServerCertThumbprint : Identifiant du certificat de notre cluster.

Précisions intéressante, Il est possible de déployer son projet sur un cluster public ! Pour effectuer un POC avec un crédit Azure restreint c'est très utile, voir : <https://try.servicefabric.azure.com/>

4.4.3.4. Communication avec Service Fabric

Pour le moment j'ai utilisé la communication synchrone par appel HTTP. A mon sens pour démarrer sur cette architecture nous pouvons commencer par cette approche. Si par la suite le besoin est présent (voir chapitre 3.3.4.2) alors une utilisation d'Azure Service Bus peut être implémenter progressivement.

Voici une méthode permettant à notre application cliente Demo.Techs.Web ou Demo.Techs.Win d'appeler un service dans Service Fabric :

```
/// <summary>
/// Lecture d'un ClientDto.
/// API déployé dans Service Fabric
/// </summary>
/// <returns></returns>
public async Task<ClientDto> LireClient()
{
    // Appel API test
    string proxyUrl = "orangems.westcentralus.cloudapp.azure.com:82/api/Clients/1";

    ClientDto result = null;
    using (HttpClient client = new HttpClient())
    {
        var response = await client.GetAsync(proxyUrl);
        if (response.IsSuccessStatusCode)
        {
            string content = await response.Content.ReadAsStringAsync();
            result = JsonConvert.DeserializeObject<ClientDto>(content);
        }
        if (!response.IsSuccessStatusCode)
        {
            throw new Exception($"{(int)response.StatusCode} - {response.StatusCode.ToString()});
        }
    }
    return result;
}
```

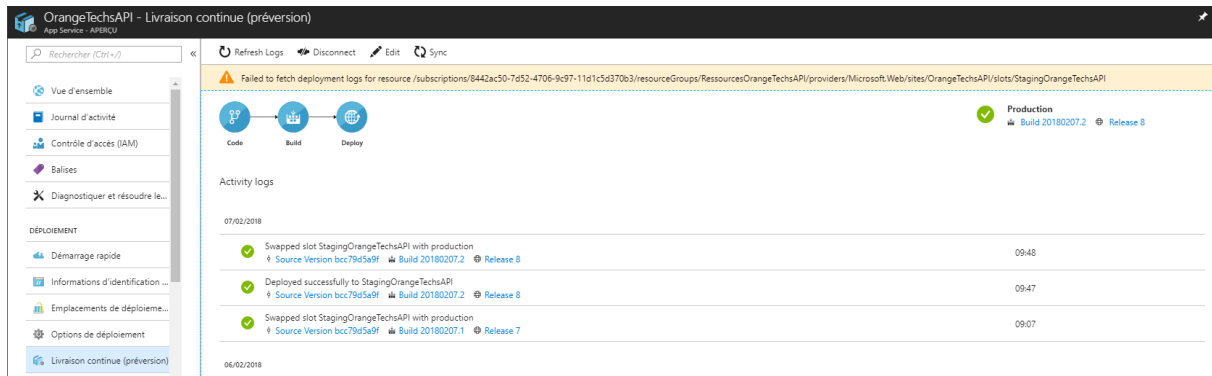
Il s'agit d'un appel d'une API REST tout à fait classique. Notre service est exposé via le port 82. Pour ce faire nous avons ouvert le port sur notre cluster lors de la création (voir ci-dessus) et configuré le port de notre service dans le fichier **ServiceManifest.xml** du projet : Verifier la **section Resources/Endpoints** du fichier.

Si un autre service de mon cluster doit réaliser le même appel, l'url sera la suivante : <http://localhost:19081/DemoTechsMicroservices/Clients/1>

On remarque que l'appel se fait sur localhost (quel que soit l'environnement) et que le port utilisé est celui du reverse proxy !

On doit également spécifier le chemin application vers notre API avec **DemoTechsMicroservices**.

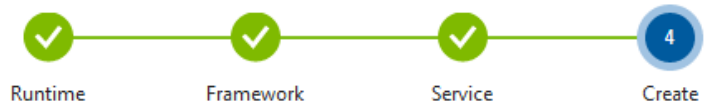
Pour notre service Demo.Techs.API j'ai créé le projet sous Visual Studio, je l'ai publié sur Azure et j'ai utilisé le module Livraison Continue de la Web Application pour la configuration dans VSTS. J'obtiens alors une configuration de build et release pour le déploiement vers Azure et je n'ai plus qu'à pousser mon code sur le dépôt créé. Cette fonctionnalité est en préversion actuellement mais semble plutôt aboutis !



J'aurais pu également utiliser Azure DevOps Project. Cette outils permet depuis Azure et sans besoin préalable de :

- Créer une Web Application du langage / framework de votre choix.
- Créer un projet et un repository git dans VSTS
- Configurer CI et CD.

Cet outils est très facile à utiliser, il suffit de quatre étapes pour générer un pipe d'intégration et de déploiement continue vers Azure.



Almost there!

Ready to deploy ASP.NET Core web app to a new Web App on Windows.

Visual Studio Team Services [Change](#)

Visual Studio Team Services (VSTS) for building and deploying your app

* Account

Create new Use existing

oranetechsdevops ✓
.visualstudio.com

* Project name

oranetechsapi ✓

Azure [Change](#)

Azure resources required for running your app

* Abonnement

Visual Studio Enterprise ▼

* App name

oranetechsdevops ✓
.azurewebsites.net

* App Service location

South Central US ▼

Pricing tier: S1 Standard

[Previous](#)

[Done](#)

Il ne reste plus qu'à récupérer le code source créer par Azure DevOps en se connectant au repository pour pouvoir commencer à travailler. Notre configuration de builds et de releases est déjà opérationnelle.

4.5. Application Lifecycle Management

Nous allons maintenant détailler la mise en place de VSTS, notre outils ALM, pour notre projet.

J'ai ensuite configurer ma build à partir du template Azure Service Fabric :

The screenshot shows the configuration page for a build process in Azure DevOps. The process is named "Orange.Techs.Microservices-Azure Service Fabric Application-Cl". It is configured to run on a "Hosted VS2017" agent. The build process includes several tasks: "Get sources" (Orange.Techs.Microservices master), "Phase 1" (Run on agent), "Use NuGet 4.4.1" (NuGet Tool Installer), "NuGet restore" (NuGet), "Build solution ***.sln" (Visual Studio Build), "Build solution ***.sproj" (Visual Studio Build), "Copy Files to: \$(build.artifactstagingdirector...)" (Copy Files), "Delete files from \$(build.artifactstagingdirec...)" (Delete Files), "Update Service Fabric Manifests (Manifest v...)" (Update Service Fabric Manifests), "Copy Files to: \$(build.artifactstagingdirector...)" (Copy Files), and "Publish Artifact: drop" (Publish Build Artifacts). The parameters section shows "Solution *" set to "***.sln" and "Service Fabric project *" set to "***.sproj".

Build Service Fabric en cours :

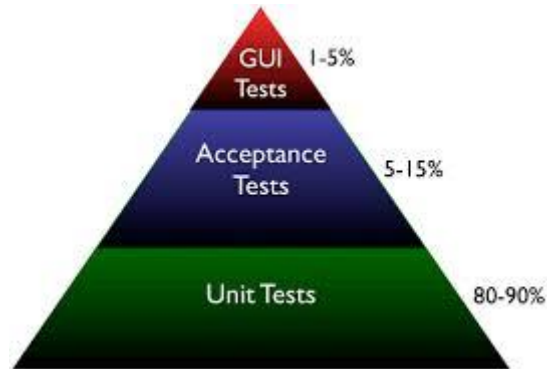
The screenshot shows the console output of the build process. The build has started and is running for 34 seconds on a Hosted Agent. The console output shows the following steps:

```
Build Started
Job
Running for 34 seconds (Hosted Agent)
Console Timeline Code coverage Tests
If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -b with the checkout command again. Example:
git checkout -b <new-branch-name>
HEAD is now at e422cd2... Correctif API
Finishing: Get Sources
Starting: Use NuGet 4.4.1
Task : NuGet Tool Installer
Description : Acquires a specific version of NuGet from the internet or the tools cache and adds it to the PATH. Use this step to change the version of NuGet used in the NuGet steps.
Version : 0.1.5
Author : Microsoft Corporation
Help : [More Information](https://go.microsoft.com/fwlink/?linkid=852538)
Downloading: https://dist.nuget.org/win-x86-commandline/v4.4.1/nuget.exe
Caching tool: NuGet 4.4.1 x64
Using Version: 4.4.1
Found tool in cache: NuGet 4.4.1 x64
Using tool path: D:\a_tool\nuget\4.4.1\x64
Prepending PATH environment variable with directory: D:\a_tool\nuget\4.4.1\x64
Finishing: Use NuGet 4.4.1
Starting: NuGet restore
Task : NuGet
Description : Restore, pack, or push NuGet packages, or run a NuGet command. Supports NuGet.org and authenticated feeds like Package Management and MyGet. Uses NuGet.exe and works with .NET Framework apps. For .NET
Version : 2.0.25
Author : Microsoft Corporation
Help : [More Information](https://go.microsoft.com/fwlink/?linkid=613747)
C:\Windows\system32\chcp.com 65001
Active code page: 65001
Detected NuGet version 4.4.1.4656 / 4.4.1
SYSTEMSSCONNECTION exists true
Saving NuGet.config to a temporary config file.
D:\a_tool\nuget\4.4.1\x64\nuget.exe sources Add -NonInteractive -Name NuGetOrg -Source https://api.nuget.org/v3/index.json -ConfigFile D:\a\1\nuget\temp\nuget_28.config
Package Source with Name: NuGetOrg added successfully.
Saving NuGet.config to a temporary config file.
D:\a_tool\nuget\4.4.1\x64\nuget.exe restore D:\a\1\src\Voting.sln -Verbosity Detailed -NonInteractive -ConfigFile D:\a\1\nuget\temp\nuget_28.config
```

Rien de particulier à signaler ici.

4.5.2. Tests

Pyramide des tests de Mike Cohn :



Types de tests :

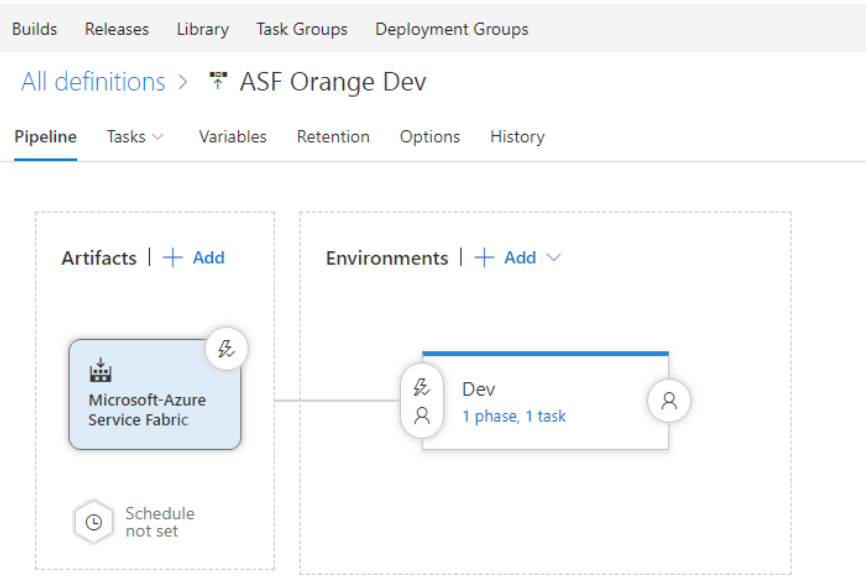
- TDD
- Tests unitaires / mock
- Services : TU / TI
- Tests d'interface
- Comment tester ASF ?
- Sonar : image

Outils tests :

- TU : MSTest V2
- Moq
- InMemoryEntityframeworkCore pour les tests d'intégration
- Soapui : tests de charge
- TestManager tests recette
- Test de performance : Soapui
- Tests d'intégration : Selenium / Coded UI
- <http://rdonack.developpez.com/tutoriels/dotnet/test-interface-utilisateur-application-asp.net-avec-selenium/>
- Parler des tests fonctionnels ? <https://blog.octo.com/les-tests-fonctionnels-en-dotnet/>

4.5.3. Déploiements

Pour le déploiement vers Azure Service Fabric c'est très simple la aussi, une seule étape VSTS suffit :



Task Deploy :

All definitions > ASF Orange Dev

Pipeline Tasks Variables Retention Options History

Dev
Deployment process

Agent phase
Run on agent

Deploy Service Fabric Application
Service Fabric Application Deployment

Service Fabric Application Deployment

Version 1,*

Display name *
Deploy Service Fabric Application

Application Package *
\$(system.defaultworkingdirectory)**/drop/applicationpackage

Cluster Connection * | Manage
ASF

Publish Profile
\$(system.defaultworkingdirectory)**/drop/projectartifacts**/PublishProfiles/Cloud.xml

Application Parameters

Advanced Settings

Upgrade Settings

Docker Settings

Control Options

Output Variables

Il faut juste configurer correctement la section Cluster Connexion :

Update Authentication for ASF ×

Certificate Based
 Azure Active Directory credential
 Others

Connection name:

Cluster Endpoint: ⓘ

Server Certificate Thumbprint(s): ⓘ

Client Certificate: ⓘ

Password: ⓘ

[Learn More](#)

Les informations demandées par cet écran sont renseignées par les infobulle. Si vous n'avez pas associé de mot de passe à votre certificat, laissez le champ vide.

Pendant l'étape de déploiement de notre projet Service Fabric, l'Explorer affichera également des informations sur le déploiement :

UPGRADE IN PROGRESS							
Current Version	Target Version	Progress by Upgrade Domain				Upgrade State	
1.0.0.20170815.3	1.0.0.20170815.4	0	1	2	3	4	RollingForwardInProgress
Show upgrade details							

La mise à jour se fait sans interruption de service puisque la mise à jour se fait par nœud. A un instant T du déploiement il y aura donc deux versions de votre application en execution simultanée. A chaque mise à jour de nœuds, Azure Service Fabric réalise des tests pour vérifier que votre mise à jour s'est bien passée. Si l'update à échoué, un rollback vers la précédente version est automatiquement réalisé.

Pour une gestion plus poussée des montée de version, on pourra mettre en place le « **Blue / Green Deployment** » qui nous permettra de valider la version à jour de notre service (par des tests sur l'interface swagger par exemple) sur un noeud avant de la déployer sur les autres.

4.6. Analyse

Détaillons maintenant les problématiques que nous avons résolus grâce à notre architecture :

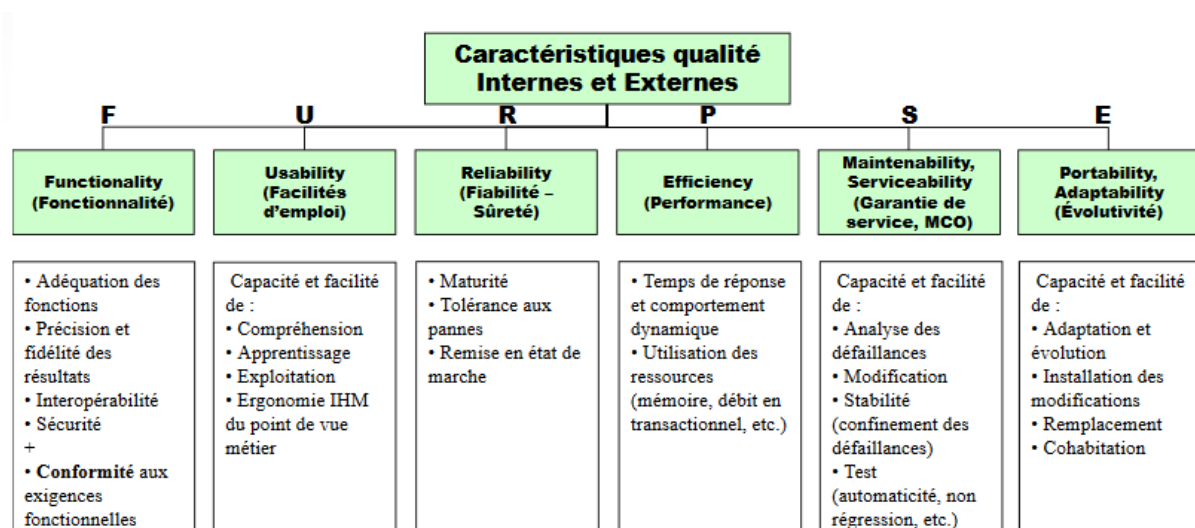
Au niveau du projet et du code .Net :

- Projet Visual Studio correctement découpé : « separation of concerns »
- Longueur des builds bien plus rapide.
- Montée en compétence plus rapide sur les projets
- Moins de couplage entre les composants.

Au niveau de l'application :

- En cas de modification, on doit tester un service.
- Déploiements plus facile et plus fréquent puisqu'ils sont réalisés sur un périmètre restreint
- La mise à l'échelle est plus simple puisqu'il est facile grâce à ASF de provisionner une nouvelle instance de mon microservice.

Reprenons notre analyse FURPSE pour détailler les avantages de notre architecture :



- **Fiabilité** : Notre application est beaucoup plus légère, et la couche de services est résiliente grâce à l'autoscaling Azure Service Fabric.
- **Performance** : Les temps de réponse sont équivalents mais la charge de calcul est répartie entre service dans le Cloud et plus sur le poste utilisateur.
- **Maintenabilité** : Le code est correctement découpé, il est donc bien plus facile à tester et debugger.
- **Évolutivité** : En cas d'évolution sur un service, on doit redéployer ce service et plus toute l'application. On peut effectuer une migration technique / technologique de façon plus simple sur un contexte réduit.

Nous avons donc répondu aux problématiques de notre existant et rendus notre application facilement déployable, testable, maintenable, évolutive et tolérante aux pannes.

5. Conclusion

Notre système d'informations de démonstration souhaitait une nouvelle application pour une utilisation en mobilité. Face aux problématiques de leur existant et les couts de développement de la nouvelle solution nous avons décidés de prendre du recul sur leur composants informatiques et de constituer une couche de service pour mutualiser leurs développements métier.

Après avoir définis différents domaines fonctionnels grâce à l'approche « Domain Driven Design » et étudié les avantages et inconvénients d'une approche microservices, nous choisissons la plateforme Azure Service Fabric. Son implémentation requière une montée en compétence bien réelle mais permet à minima au développeurs de conserver leurs abitudes lors du développement d'API ASP .NET Core.

Certes nous avons compléxifier le système d'information dans sa globalité car nous avons plus de composants impliquant déploiement et surveillance, mais en contrepartie celui-ci est devenu bien plus évolutif face aux besoins interne comme externe de l'entreprise.

Azure Service Fabric dans son rôle d'orchestration de services réponds parfaitement à cette implémentation.

Annexe A. Articles et tutoriels utilisés

<https://stackoverflow.com/questions/30213456/transactions-across-rest-microservices>

<https://docs.microsoft.com/fr-fr/azure/service-fabric/service-fabric-cloud-services-migration-differences>

<https://docs.microsoft.com/fr-fr/azure/service-fabric/service-fabric-api-management-overview>

<https://blog.octo.com/designer-une-api-rest/>

<https://docs.microsoft.com/en-us/azure/service-fabric/service-fabric-overview>

<https://docs.microsoft.com/fr-fr/dotnet/standard/microservices-architecture/architect-microservice-container-applications/asynchronous-message-based-communication>

<https://docs.microsoft.com/fr-fr/dotnet/standard/microservices-architecture/architect-microservice-container-applications/direct-client-to-microservice-communication-versus-the-api-gateway-pattern>

<https://docs.microsoft.com/fr-fr/dotnet/standard/microservices-architecture/architect-microservice-container-applications/communication-in-microservice-architecture>

<https://docs.microsoft.com/fr-fr/azure/architecture/microservices/interservice-communication>

<http://blogashwani.blogspot.fr/2016/09/api-gateway-with-azure-service-fabric.html>

<https://plainoldobjects.com/2015/09/02/does-each-microservice-really-need-its-own-database-2/>

<http://blog.soat.fr/2015/06/azure-service-fabric-le-paas-v2-de-microsoft>

<https://www.codeproject.com/Articles/1217885/Azure-Service-Fabric-demo>

<https://www.developpez.com/actu/193669/DevOPS-avec-Azure-moins-Partie-8-a-la-decouverte-de-l-outil-App-Service-Continuous-Delivery-par-Hinault-Romarc/>

<https://docs.microsoft.com/fr-fr/azure/architecture/microservices/data-considerations>

<https://docs.microsoft.com/fr-fr/azure/service-fabric/service-fabric-reliable-services-communication-aspnetcore>

<https://docs.microsoft.com/fr-fr/dotnet/standard/microservices-architecture/architect-microservice-container-applications/scalable-available-multi-container-microservice-applications>

<https://martinfowler.com/articles/microservices.html>

https://cdn2.hubspot.net/hubfs/2762090/documents/Ebook_guide_dev_Azure_dev_premier_jour_fr.pdf?t=1518449020200&utm_campaign=Digidev%20-%20consideration&utm_source=hs_automation&utm_medium=email&utm_content=59447710&hsenc=p2ANqtz-_KARcoEWzG4bbhN7xm_DzqR2gvByeVQMIrEagrUMIRLOWyPwYI5YwsMtJ2FYHgmInKNoaaDGJy8rQaTPz2gfJ-Dlwuw&_hsmi=59447710

<https://docs.microsoft.com/fr-fr/azure/service-fabric/service-fabric-connect-to-secure-cluster>

<https://docs.microsoft.com/fr-fr/azure/service-fabric/service-fabric-cluster-creation-via-portal>

<http://opikanoba.org/tr/fielding/rest/>

<https://docs.microsoft.com/fr-fr/azure/architecture/resiliency/index>

<https://docs.microsoft.com/fr-fr/azure/service-fabric/service-fabric-visualstudio-configure-secure-connections>

<http://www.khamis.net/2017/02/03/choosing-between-azure-container-services-azure-service-fabric-and-azure-functions/>

<https://12factor.net/>

<https://docs.microsoft.com/fr-fr/azure/service-fabric/service-fabric-containers-overview>

<https://docs.microsoft.com/fr-fr/azure/service-bus-messaging/service-bus-azure-and-service-bus-queues-compared-contrasted>

<https://docs.microsoft.com/fr-fr/azure/architecture/microservices/>

https://www.comsoft-dev.fr/hubfs/documents/LB_Prez_Azure_guide_usage_developpeur_eBook_fr.pdf?t=1518449020200&utm_campaign=Digidev%20-%20consideration&utm_source=hs_automation&utm_medium=email&utm_content=59638702&hsenc=p2ANqtz--CjGmBEaWQf5VFDIGSG4ot7SpFoStO3iecg8LDMWaRfuweEwX8gTEKvUhy7uU3U21tyiauQoIH6TvCeYccJQkASi_pQ&_hsmi=59638702

<https://docs.microsoft.com/en-us/vsts/user-guide/connect-team-projects>

<http://blog.xebia.fr/2015/03/09/microservices-des-architectures/>

<https://azure.microsoft.com/fr-fr/services/service-fabric/>

<http://rdonfack.developpez.com/tutoriels/dotnet/nouveautes-aspnet-core-2/>

<https://blog.octo.com/larchitecture-microservices-sans-la-hype-quest-ce-que-cest-a-quoi-ca-sert-est-ce-quil-men-faut/>

<https://blogs.msdn.microsoft.com/maheshkshirsagar/2016/11/21/choosing-between-azure-container-service-azure-service-fabric-and-azure-functions/>

<https://stackoverflow.com/questions/41043912/unable-to-authenticate-fabricclient-to-a-secured-service-fabric-cluster>

<https://docs.microsoft.com/en-us/vsts/git/tutorial/gitworkflow>

<https://docs.microsoft.com/en-us/vsts/git/gitquickstart?tabs=visual-studio>

<https://blog.octo.com/strategie-d-architecture-api/>

<https://fr.slideshare.net/baronm/introduction-aux-architectures-microservices-introduction-gnrale>

<http://blog.xebia.fr/2015/03/09/microservices-des-architectures/>

<http://blog.xebia.fr/2015/03/16/microservices-des-pieges/>

<http://www.it-expertise.com/architectures-micro-services-objectifs-benefices-et-defis/>

<http://blog.xebia.fr/2015/03/02/microservices-les-concepts/>

<https://www.technologies-ebusiness.com/enjeux-et-tendances/architectures-micro-services-objectifs-benefices-defis-partie-2>

<https://stackoverflow.com/questions/41834111/azure-service-fabric-vs-azure-container-services>

<https://www.developpez.com/actu/149264/ASP-NET-Core-moins-Apprendre-a-utiliser-les-Tag-Helpers-un-billet-d-Hinaul-Romarc/>

<https://blogs.msdn.microsoft.com/maheshkshirsagar/2016/11/21/choosing-between-azure-container-service-azure-service-fabric-and-azure-functions/>

<http://www.gaunacode.com/azure/service-fabric/containers/2017/04/19/ServiceFabric-vs-AzureContainerServices.html>

<http://www.khamis.net/2017/02/03/choosing-between-azure-container-services-azure-service-fabric-and-azure-functions/>

<https://docs.microsoft.com/fr-fr/azure/service-fabric/service-fabric-containers-overview>

<https://www.softfluent.fr/blog/expertise/2015/08/31/Comment-documenter-son-service-REST-avec-ASP-NET-Web-API>

<https://www.developpez.net/forums/blogs/137868-hinault-romaric/b2531/deploiement-application-asp-net-core-conteneur-docker/>

<https://www.developpez.com/actu/145896/-NET-Core-ou-NET-Framework-Quelle-implementation-adopter-pour-son-projet-par-Hinault-Romaric/>

<http://www.supinfo.com/articles/single/3458-creer-une-api-c-avec-web-api-entity-framework-6>

<https://www.developpez.com/actu/148669/Conteneuriser-son-application-NET-quel-OS-hote-utiliser-un-billet-d-Hinault-Romaric/>

<https://stackoverflow.com/questions/41834111/azure-service-fabric-vs-azure-container-services>

<http://soat.developpez.com/tutoriels/web/conditions-utilisation-microservices/>

<https://www.developpez.net/forums/blogs/137868-hinault-romaric/b2002/azure-web-apps-slot-deploiement-livraison-continue-visual-studio-team-services/>

<https://www.developpez.com/actu/148669/Conteneuriser-son-application-NET-quel-OS-hote-utiliser-un-billet-d-Hinault-Romaric/>

<http://blog.soat.fr/2015/06/azure-service-fabric-le-paas-v2-de-microsoft>

<https://docs.microsoft.com/fr-fr/azure/service-fabric/service-fabric-overview-microservices>

<https://anthonychu.ca/post/aspnet-core-service-fabric-party-cluster/>

<http://www.christian-faure.net/2008/10/08/le-style-darchitecture-soa/>

<https://stackoverflow.com/questions/35613841/most-efficient-way-to-communicate-between-multiple-net-apps?answertab=oldest#tab-top>

<https://www.christopheducamp.com/2013/12/16/github-pour-nuls-partie-2/>

<https://martinfowler.com/>

<https://martinfowler.com/>

<http://rogerdudler.github.io/git-guide/index.fr.html>

<https://www.developpez.com/actu/179848/Aller-plus-loin-avec-la-configuration-de-Swagger-dans-une-Web-API-ASP-NET-Core-par-Hinault-Romaric/>

<http://rdonfack.developpez.com/tutoriels/documenter-web-api-aspnet-core-swagger/>

<http://www.supinfo.com/articles/single/503-creer-une-api-restful-avec-aspnet>

<http://rdonfack.developpez.com/tutoriels/alm/devops-avec-vsts-azure-aspcore>

<https://docs.microsoft.com/fr-fr/vsts/build-release/archive/apps/aspnet/aspnet-4-ci-cd-azure-automatic>

<https://docs.microsoft.com/fr-fr/dotnet/standard/microservices-architecture/multi-container-microservice-net-applications/data-driven-crud-microservice>

<http://deepin.developpez.com/articles/asp-net-web-api/creation-api-web-asp-net-sur-azure-web-sites/>

<http://rdonfack.developpez.com/tutoriels/dotnet/decouverte-asp-net-web-api>

<http://rdonfack.developpez.com/tutoriels/dotnet/creation-application-aspnetcore-sous-linux/#LX>

<https://dotnet.developpez.com/actu/110983/Livraison-continue-dans-un-projet-ASP-Net-avec-les-fonctionnalites-Release-Management-dans-Visual-Studio-Team-Services-un-tutoriel-de-Hinault-Romarc/>

<https://dotnet.developpez.com/actu/105696/MsTest-V2-tests-unitaires-parametriques-un-tutoriel-de-Hinault-Romarc/>

<https://dotnet.developpez.com/actu/110229/Integration-continue-d-un-projet-ASP-NET-Core-avec-Visual-Studio-Team-Services-un-tutoriel-de-Hinault-Romarc/>

<http://blog.cellenza.com/cloud-2/azure/prise-en-main-des-azure-api-apps-au-coeur-de-larchitecture-microservices/>

<https://www.developpez.com/actu/103829/Apprendre-a-creer-une-application-CRUD-avec-ASP-NET-Core-et-Entity-Framework-Core-un-tutoriel-de-Hinault-Romarc/>

<https://docs.microsoft.com/fr-fr/dotnet/csharp/tutorials/microservices>

<https://www.developpez.com/actu/166144/DevOps-avec-Azure-moins-Partie-3-apprendre-a-creer-et-a-deployer-des-ressources-avec-Visual-Studio-et-ARM-Template-par-Hinault-Romarc/>

<https://docs.microsoft.com/fr-fr/azure/service-fabric/service-fabric-get-started-azure-cluster>

<https://docs.microsoft.com/fr-fr/azure/service-fabric/service-fabric-quickstart-dotnet>

<http://rdonfack.developpez.com/tutoriels/asp-net-core-razor-pages/>

<https://dotnetthoughts.net/creating-your-first-aspnet-core-web-api-with-swashbuckle/>

<https://stackoverflow.com/questions/41043912/unable-to-authenticate-fabricclient-to-a-secured-service-fabric-cluster>

<https://docs.microsoft.com/fr-fr/azure/app-service/app-service-web-get-started-dotnet>

<https://docs.microsoft.com/fr-fr/azure/service-fabric/service-fabric-tutorial-create-dotnet-app>

<https://docs.microsoft.com/fr-fr/azure/service-fabric/service-fabric-tutorial-deploy-app-to-party-cluster>

<https://www.developpez.net/forums/blogs/137868-hinault-romaric/b3361/asp-net-core-introduction-aux-tag-helpers/>

<https://www.developpez.com/actu/154398/Apprendre-l-injection-de-dependances-avec-ASP-NET-Core-un-billet-d-Hinault-Romaric/>

<http://rdonfack.developpez.com/tutoriels/dotnet/asp.net-core-mise-en-place-tests-unitaires-application-mvc/#LV-A>

<https://www.developpez.com/actu/138334/Demystifier-le-modele-MVC-des-applications-ASP-NET-Core-moins-partie-4-regroupement-minification-et-CDN-par-Hinault-Romaric/>

<https://cdiese.fr//conception-microservices>

<https://docs.microsoft.com/fr-fr/azure/service-fabric/service-fabric-connect-and-communicate-with-services>

<https://docs.microsoft.com/fr-fr/azure/architecture/guide/architecture-styles/microservices>

<https://docs.microsoft.com/fr-fr/azure/service-fabric/service-fabric-reverseproxy>

<https://stackoverflow.com/questions/30213456/transactions-across-rest-microservices>

<https://compagnondevoyage.fr/editorial/technique/cqrs-es/>

<https://docs.microsoft.com/fr-fr/azure/application-insights/app-insights-usage-overview>

<https://docs.microsoft.com/fr-fr/azure/application-insights/app-insights-detect-triage-diagnose>

Annexe B. Vidéos

<https://channel9.msdn.com/Events/Build/2017/B8106>

<https://channel9.msdn.com/Events/Build/2017/T6051>

<https://channel9.msdn.com/Events/Build/2017/P4020>

<https://www.youtube.com/watch?v=2yko4TbC8cl&>

<https://www.youtube.com/watch?v=PjUsN6TU-3w>

<https://www.youtube.com/watch?v=DxVxGgSnoBw>

Annexe C. Livres

ARCHITECTURE LOGICIELLE: CONCEVOIR DES APPLICATIONS SIMPLES, SURES ET ADAPTABLES. Jacques Printz

PRATIQUE DES TESTS LOGICIELS. J.F Pradat-Peyre, Jacques Printz

.NET MICROSERVICES - ARCHITECTURE FOR CONTAINERIZED .NET APPLICATIONS :
<https://blogs.msdn.microsoft.com/cesardelatorre/2017/05/10/free-ebookguide-on-net-microservices-architecture-for-containerized-net-applications/>

CLOUD APPLICATION ARCHITECTURE GUIDE : http://info.microsoft.com/rs/157-GOE-382/images/EN-CNTNT-Whitepaper-CloudApplicationArchitectureGuide.pdf?wt.mc_id=azurebg_ENAzureNewsletter_December&mkt_tok=eyJpIjoiWVdRMk1UTXdOak14ToRZMYlslNQiOil3VENcL2R4bUFyODVZR3NTWTZKQnFJOUooYXNVSHFsV2h3XC9oVmVRa1wveVY3eFcwdVFTVDRBT3VoZjZxWG9hNWN5YjRMZXV4TXZhT2NSSkp3NEdNNElrd2VOXC9ZUDJUVGFkanNnQVRCSHpodm9rdVBCb3RsakZBXC81eDJoRXRuWHNyMGk1Uzd2SkNsMDM2V3ErckhMcGooOWdDSofDZUNPblxTVwvaTVHbWRVTms9Ino%3D