

Les Fondements Des Microservices

Auteur: CAMARA Laby Damaro



Développeur Full Stack & netpreneur

Ressource: [LinkedLearning](#)

“Ma Promesse est le sommeil & la mort”

Résumé De La Formation

Les microservices font partie des modèles architecturaux de référence dans le secteur des logiciels. Il est essentiel d'avoir une vue d'ensemble de ce qu'est cette architecture et de ce qu'elle n'est pas pour se lancer dans l'évaluation de ce modèle. Cette formation couvre les concepts de base des microservices, afin de vous aider à déterminer si ce modèle architectural vous convient, à vous et à votre équipe. Par la suite, elle vous aide à vous familiariser avec certains concepts clés des microservices, y compris la délimitation des contextes et la couche d'API. Elle évoque aussi certains aspects plus avancés de l'architecture, tout en insistant sur l'importance d'adopter une culture DevOps pour migrer vers les microservices.

Programme De La Formation

Aperçu Historique Des Microservices

- 1.** Architecture monolithique n-tier
- 2.** Architecture orientée services (SOA)
- 3.** Microservices

Concepts Clés Des Microservices

- 1.** Services
- 2.** Communications
- 3.** Distribution
- 4.** Base de données et transactions
- 5.** Couche d'API

Concepts Avancés

- 1.** Communications asynchrones
- 2.** Traçage et journalisation
- 3.** Livraison continue
- 4.** Architectures hybrides

Choix En Matière D'Architecture

- 1.** Conception
- 2.** Gestion des compromis
- 3.** Services Edge
- 4.** Culture DevOps

Etablir Les Prérequis

Managers

Que vous soyez managers ou décideurs dans le cycle de vie d'une équipe de développement logiciel, nous allons dans cette formation:

- Présenter les avantages
- Présenter les inconvénients
- Mettre en avant l'impact sur les résultats

Architectes

Si vous êtes architectes cette formation constituera un excellent point de départ et vous permettra d'acquérir des bases solides pour prendre des décisions:

- Constituer un point de départ
- Poser les bonnes questions
- Découvrir les avantages et les défis

Développeurs

Si vous êtes développeurs, cette formation vous donnera de très bonnes bases pour expliquer pourquoi vous codez comme vous le faites:

- Constituer une base
- Répondre aux questions
- Ne pas se focaliser sur le code

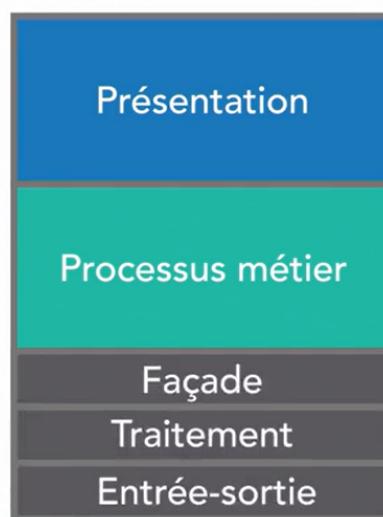
Pour suivre cette formation:

- il faut simplement avoir envie d'apprendre
- connaître les base du développement logiciel
- en tant que développeur ou architectes, vous devez connaître la composition et la décomposition logiciel
- vous devez comprendre les principes de la communication à distance via le protocole **HTTP** et **HTTPS** afin de cerner certaines des complexités des architectures **microservices**

Explorer L'Historique Des Architectures Basées Sur Les Services

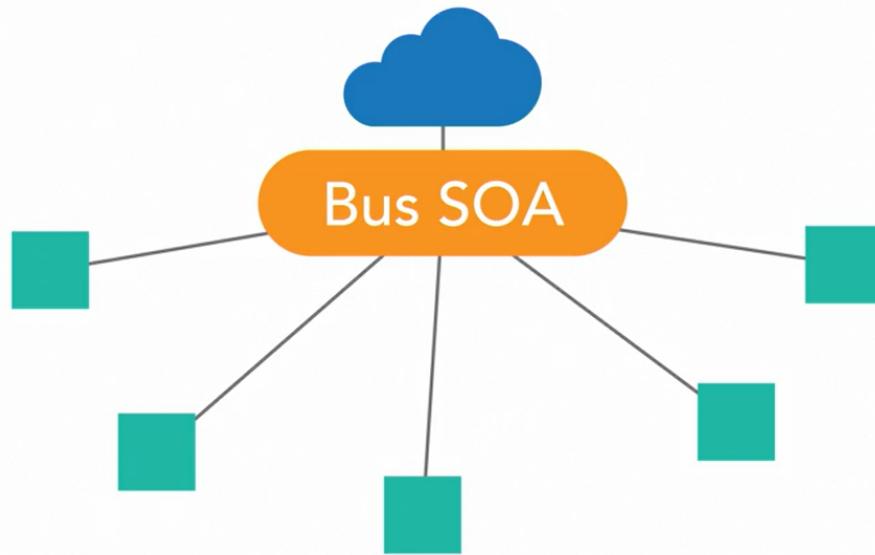
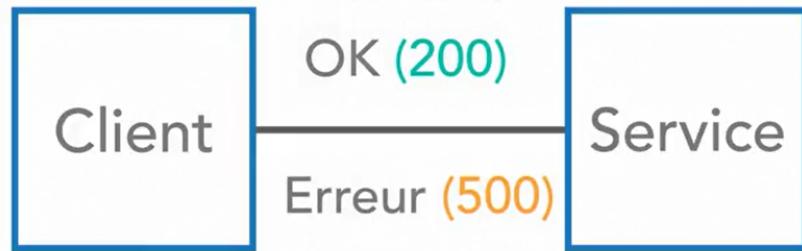
Architecture Monolithique

Avec les architectures monolithique, il faut créer un artefact constitué en trois (3) couches en générales(Présentation, Processus métier, Accès aux données), souvent chaque couche se décompose en des sous couches par exemple pour la couche **accès aux données**(Façade, traitement, Entrée-sortie), cela permet de séparer le problème et de décomposer le code en composant fonctionnel. Autre que ces problème avec les applications monolithiques il y a un problème de couplage fort, leurs conceptions prennent assez de temps et le test peut paraître parfois trop pénible.



Architecture De Service

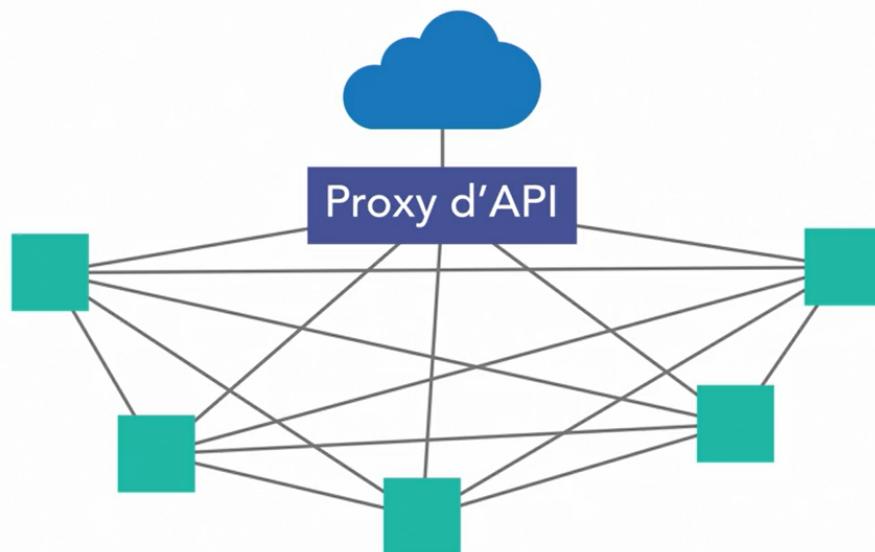
Puis nous avons aussi l'avènement des architectures de service notamment des architectures orientées services(SOA) qui permettent de décomposer les applications en plus petits modules mais génèrent des nouveaux problèmes la technologie de service web repose sur le protocole SOAP ce qui provoque de nouveaux problèmes, chaque réponse est 200(OK) ou 500(Erreur interne) en plus le problème de couche agrégation car les transformation du XML et les opérations logiques étaient rajoutées aux **Bus SOA** cela crée un nouveau niveau de couplage cependant la **SOA** s'était imposée à un moment donné de l'histoire.



Microservices

Les microservices offrent un Framework plus agile qui peut être étendu dans un environnement Cloud Natif beaucoup plus facilement que les applications monolithique ou SOA.

Ce modèle convient mieux aux développeur web et aux développeurs des services web car ils reposent sur le protocole **HTTP**



Microservices

Définition

Les microservices constituent la décomposition d'un problème logiciel en plus petits problèmes, plus facile à comprendre et à résoudre, mais ils servent aussi à faire en sorte que l'architecture au niveau des composants fonctionne de la même manière que les bonnes pratiques de développement avec la modularisation du code. Face un problème logiciel, nous savons qu'il faut décomposer le problème en plus petits problèmes pour résoudre chacun de ces problèmes de façon modulaire et découpé. Les microservices reprennent ce problèmes et l'appliquent à l'ensemble du système.

Décomposition

Décomposer un problème logiciel en plus petits problèmes, plus faciles à comprendre et à résoudre

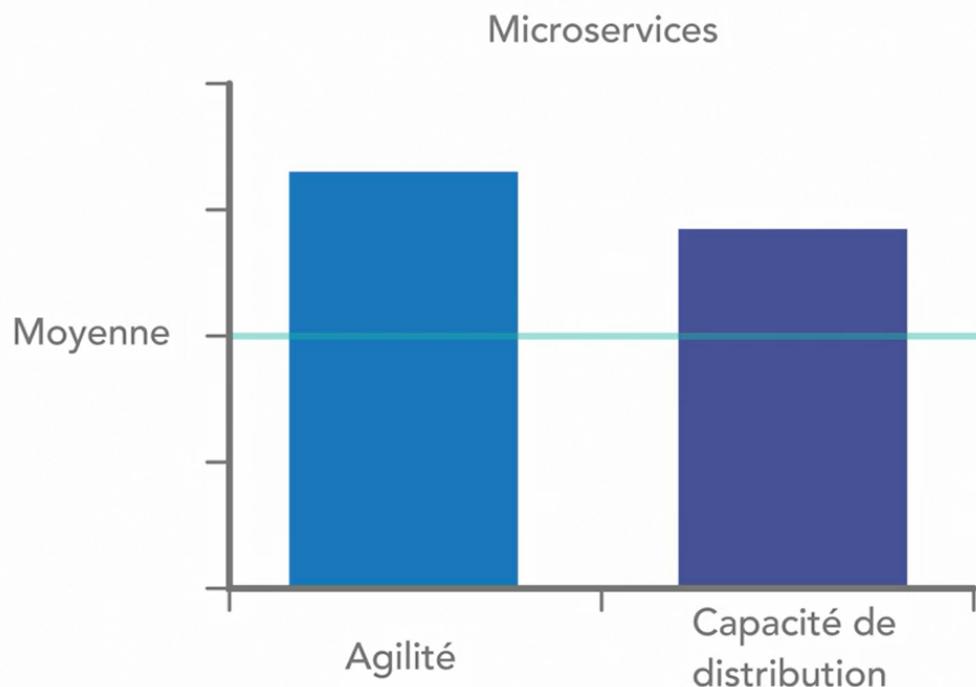
Décomposition d'un système

Décomposer un système en plus petites unités, qui facilitent la résolution de divers problèmes système

Les microservices utilisent l'interopérabilité hétérogène basée sur l'adaptation au protocole pour gérer les communications ainsi:

Tous les appels sont résolus via REST

Prise en charge du développement polyglotte



LinkedIn Learning

Distinguer Microservice Et Cloud Natif

Architecture Cloud Natif

Les architectures Cloud Nativs reposent sur la méthodologie de création d'application à 12 facteurs.

Les architectures cloud natifs incluent des modèles pour créer des systèmes s'exécutant dans une infrastructure cloud.

Infrastructure Cloud

les infrastructure cloud peuvent être:

- **Cloud public**
- **Cloud privé**
- **Cloud hybride**

Le cloud computing est un modèle de système distribuer à l'échelle mondiale qui maximise la disponibilité, évolutivité et distribution. Nous pouvons créer des systèmes cloud s'exécutant dans un seul data center et prévoir de les étendre.

Les microservices s'intègrent très bien trop souvent dans une architecture cloud natif car ils constituent une transition très fluide vers une application à 12 facteurs. ils sont deux facteurs très différents.

Nous pouvons créer des applications cloud monolithique et des application microservices incapables de migrer dans le cloud.

Trop souvent les gens confondent les deux modèles.

Mais malgré leurs différences ces deux modèles vont souvent de pairs

Applications De Microservices Cloud Natives

- Codebase unique
- Unité autonome
- Pas d'utilisation des fichiers système

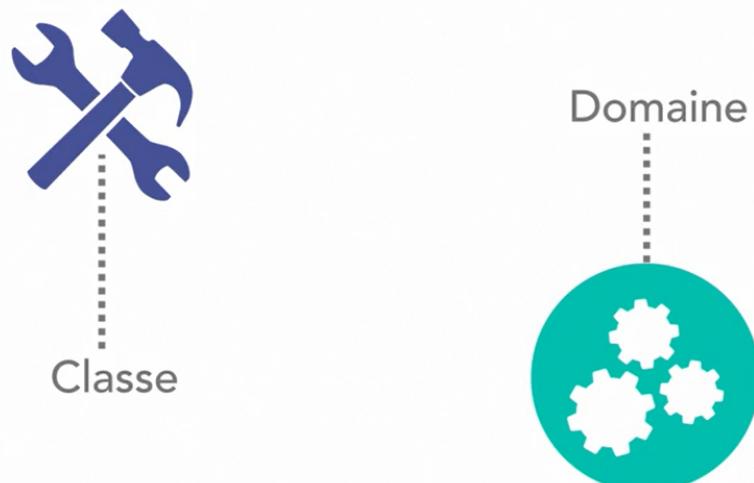
Différencier Les Services

Avec les microservices toutes les comminucations entre les services fonctionnent avec le REST à l'aide du protocole HTTP, il y a d'autre méthode base sur la notion d'évènement, mais les appels intra-service repose tous sur le REST.

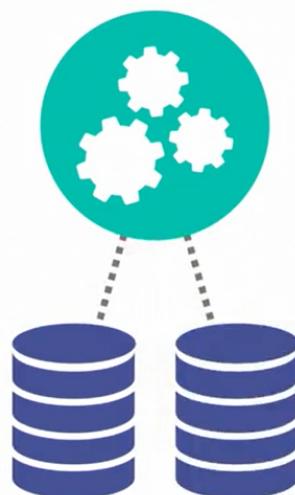


Service

Avec les microservices, un service fonctionne sur un domaine bien défini, les opérations sont définis sur le domaine et non sur les objets métiers.



Dans une architecture de microservice il faut fournir des services de très bon niveaux, souvent ses services exposent des opérations GRUD spécifique à un domaine sur l'objet de domaine, mais ce domaine peut couvrir plusieurs domaines.



Blocage De Conception De Service

- 1.** Trop affinés
- 2.** Pas assez affinés

Aborder La Danse De La Communication

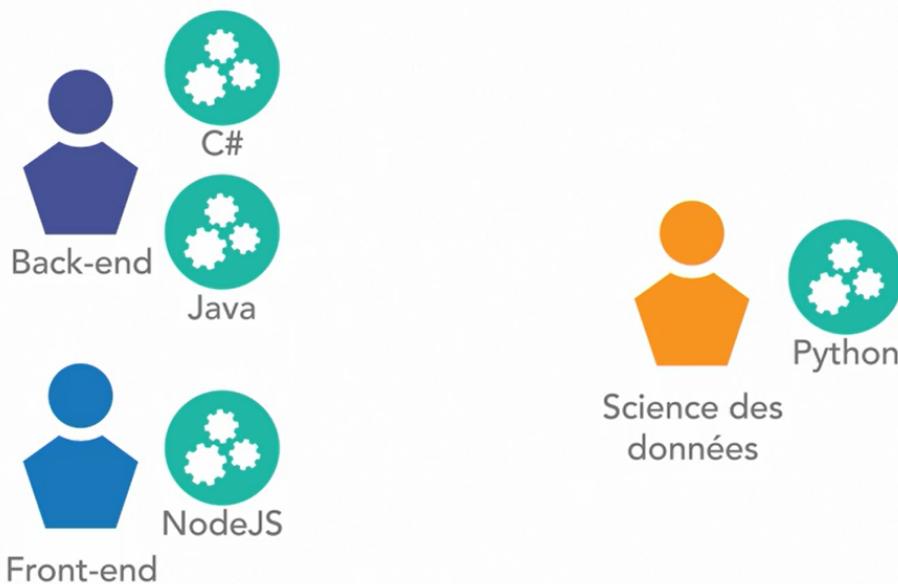
Toutes les communications entre services individuels sont effectuées via HTTP à l'aide de services REST.

Permet d'utiliser n'importe quel langage ou framework de code prenant en charge les services RESTful.

Interopérabilité Hétérogène Basée Sur L'Adaptation Aux Protocoles

Les services sont liés à un protocole et exécutent la communication via ce protocole, le tout d'une manière qui fonctionne dans un environnement mixte ou hétérogène.

Avantages



Problème

Des problèmes peuvent survenir lors de tous ces appels réseau, car chaque service peut appeler n'importe quel autre service.

Solution

Chaque service doit maintenir un certain niveau de passivité dans ces API pour éviter les défaillances dans ces systèmes, un service pouvant appeler n'importe lequel de ces services, il n'y a pas de délimitation claire qui vous appelle, nous devons avoir une solide stratégie de gestion de version de API passive pour que les systèmes d'appel ne tombent pas en panne lors de la publication d'une nouvelle version d'API.



Examiner La Distribution Et L'Evolutivité



Besoins Liés à Internet Du Point De Vue De L'Entreprise

- Applications destinées au client
- Applications destinées à l'entreprise
- Capacité de distribution dès le départ

Avantage

- évolutivité
- distribution
- scalabilité

Anticiper Les Danger De La Latence Et L'Engorgement

La Latence

La latence peut devenir intolérable dans une architecture microservice



Appels Circulaire

Les appels circulaire peuvent devenir problématiques.



Avant de passer à une architecture nous devons penser aux problèmes de latence.

Définir Le Contexte Délimité

Modèle De Conception

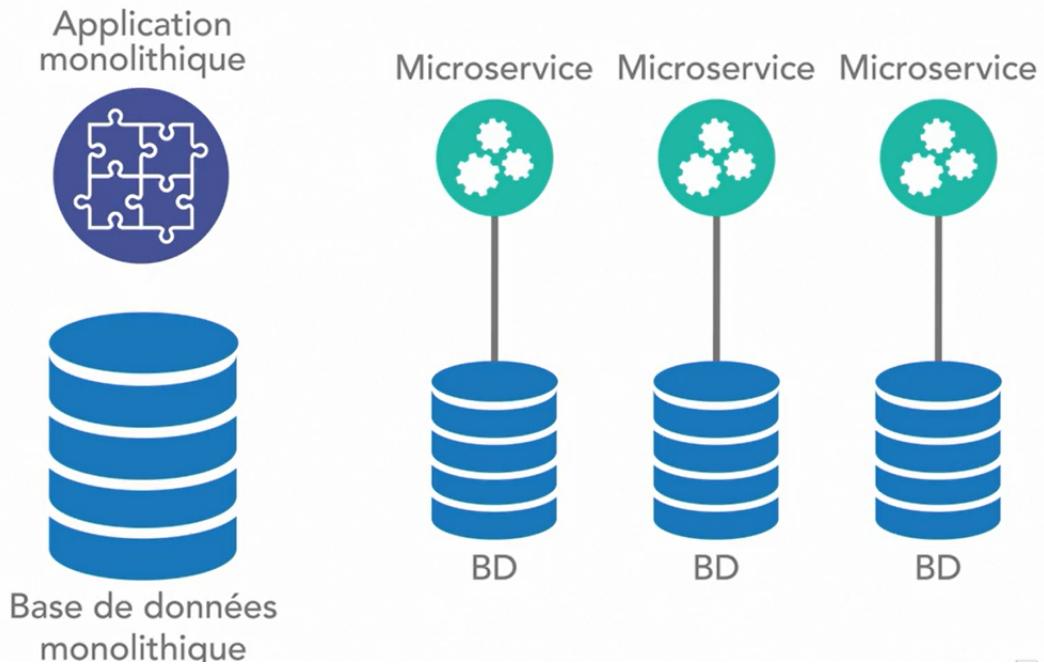
- Examiner le système de travail
- Déterminer les domaines
- Décomposer les services en conséquence

En optimisant les contrats et en délimitant précisément vos contextes, vous aboutissez à une auto-découverte de votre système.

Observer Les Limites Des Domaines De Données

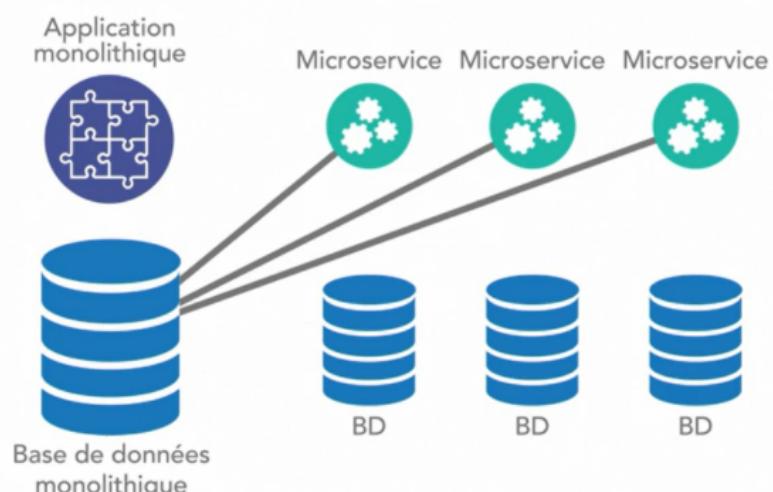
Limites Transactionnelles

- Impossible de supprimer complètement les transactions
- Pas de transactions distribuées
- Nécessité de trouver de nouvelles façons de penser



Si vous travaillez sur une couche d'API

Couche d'API



Découvrir Les Pertes Des Transactions Atomiques

Atomique **C**ohérent **I**solé **D**urable **A**tomique

Peut soit réussir complètement, soit échouer complètement

Cohérente

Garantit l'application de toutes les contraintes du modèle de données

Isolée

Impossibilité pour une autre transaction de lire des données dont l'état n'est pas correct, à cause des règles de visibilité

Durable

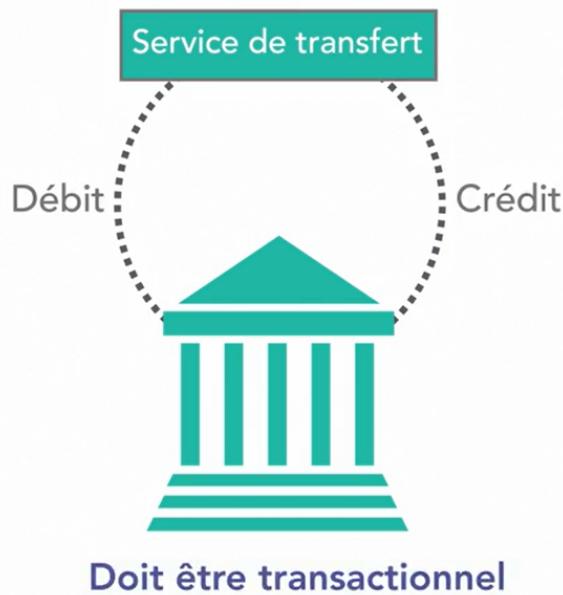
Une fois terminée, est garantie dans la banque de données jusqu'à être modifiée

Dans une architecture monolithique ces opérations bien respectés par contre dans une architecture microservice, ces opérations sont presque inexistantes à la place:

Nous cherchons à obtenir des transactions BASE et nous nous efforçons d'assurer une cohérence éventuelle.

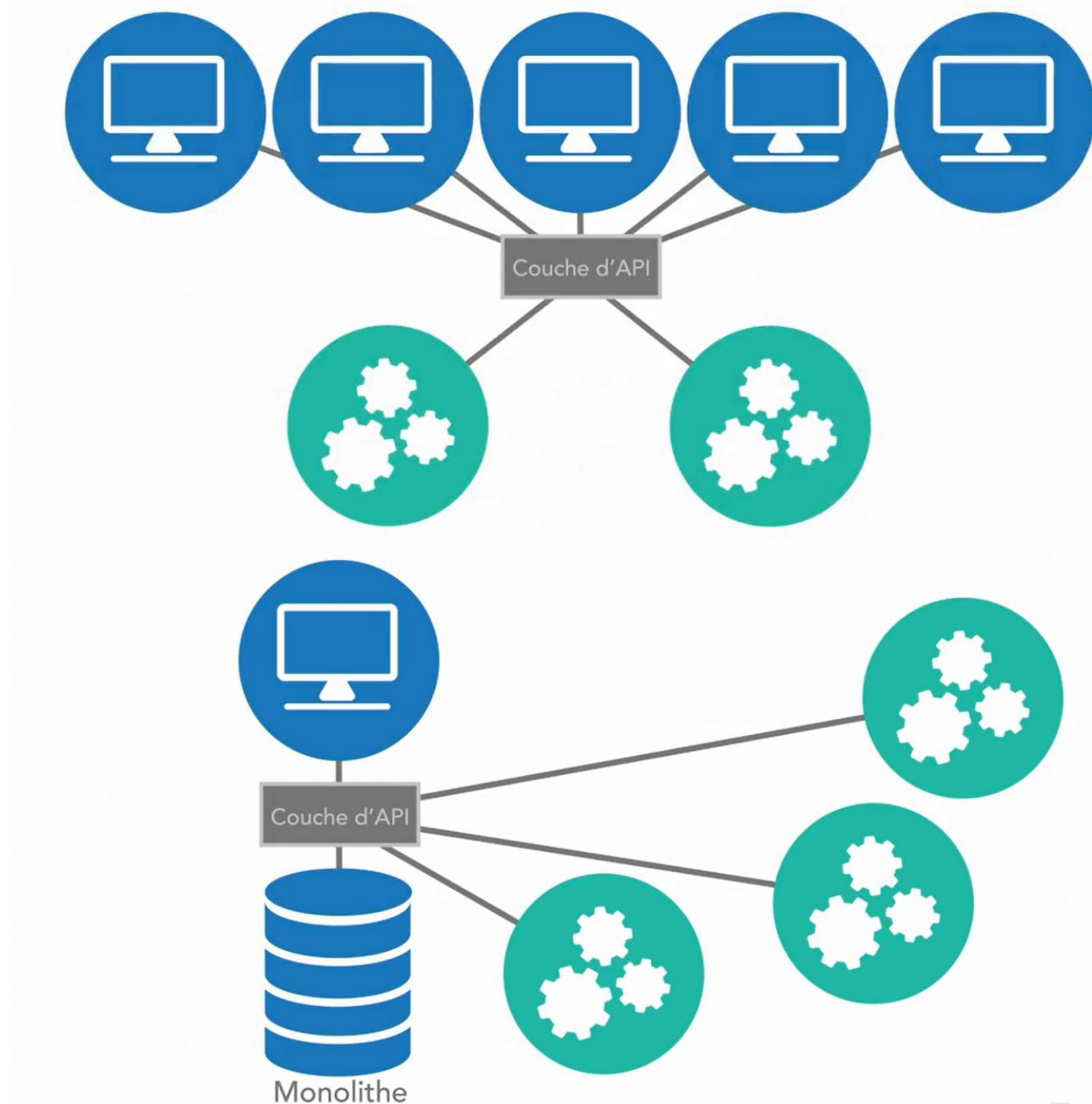
Nous atteignons l'état final dans tous les nœuds de la banque de données distribuée.

Mais il y a des moments où nous avons réellement besoin des transactions ACID par exemple pour les applications bancaires où les débits et les crédits doivent se produire dans une seule transaction bancaire.

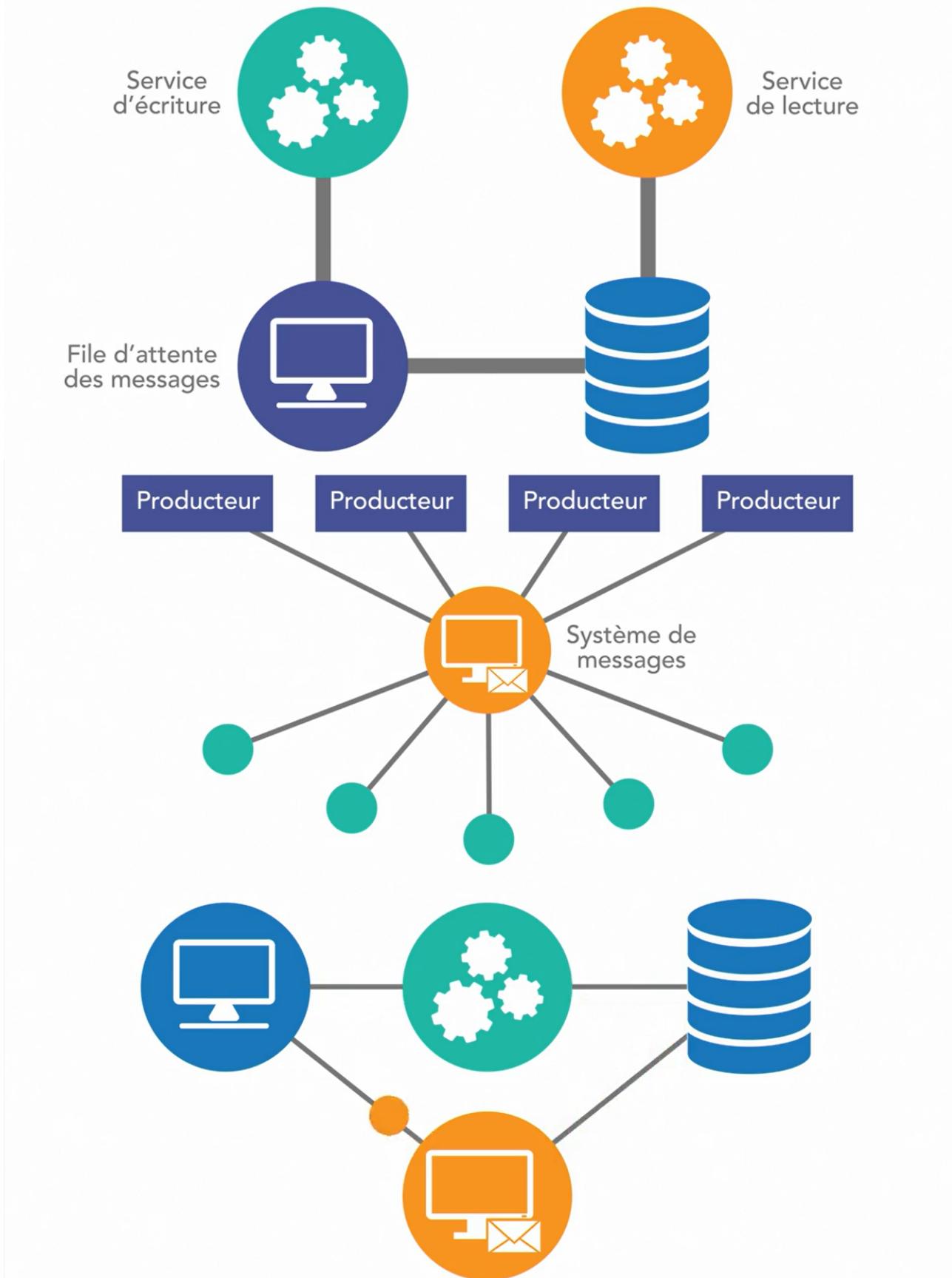


Etudier La Couche API

La couche API est un proxy agrégant toutes nos offres de service



Aborder Les Communications Asynchrones



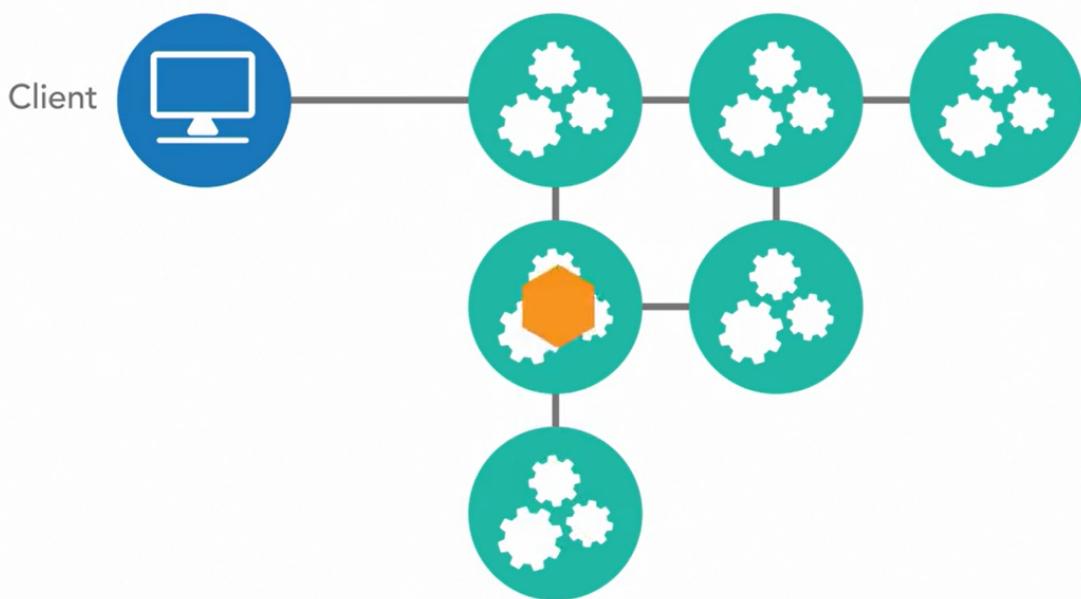
Enregistrer Et Tracer Dans Une Architecture De Microservices

- Opérations quotidiennes
- Dépannage
- Maintenance
- Investigations
- Tâches générales

La journalisation peut devenir beaucoup plus complexe dans un environnement de microservices.

- Plus grand nombre d'artefacts
- Agilité
- Différentes équipes, différentes stratégies de journalisation

Le traçage consiste à créer un jeton unique appeler trace à utiliser cette trace dans tous les événements de journalisation interne cette pile d'appelle en intégrant cette valeur à tous les résultats de journalisation et de minutage pour tous les services concernés pour faciliter la journalisation.

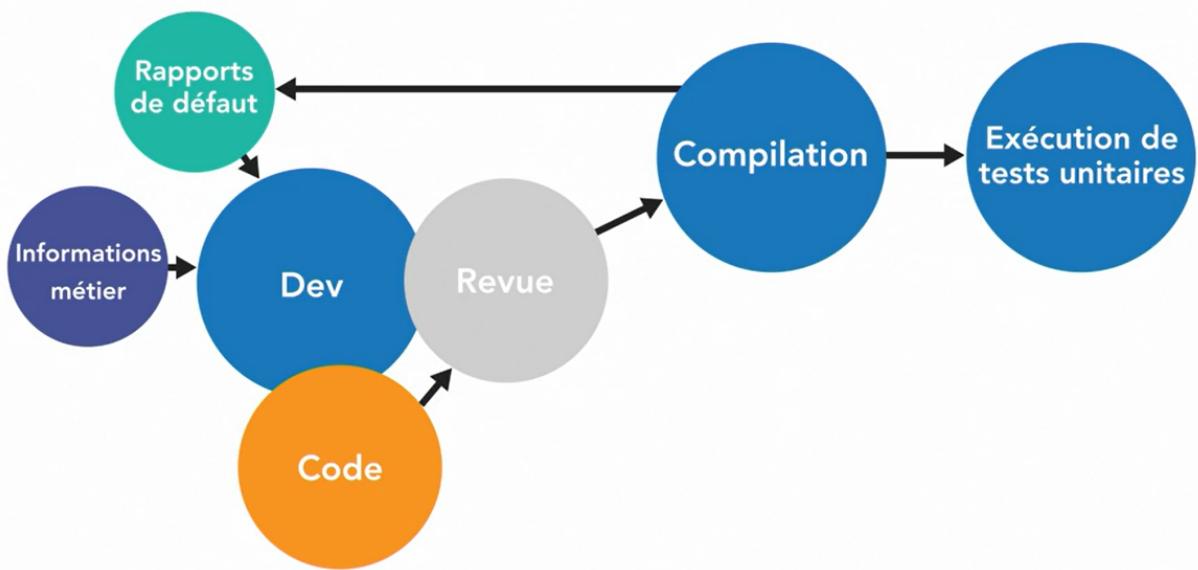


Comprendre L'Importance De La Livraison Livraison Continue

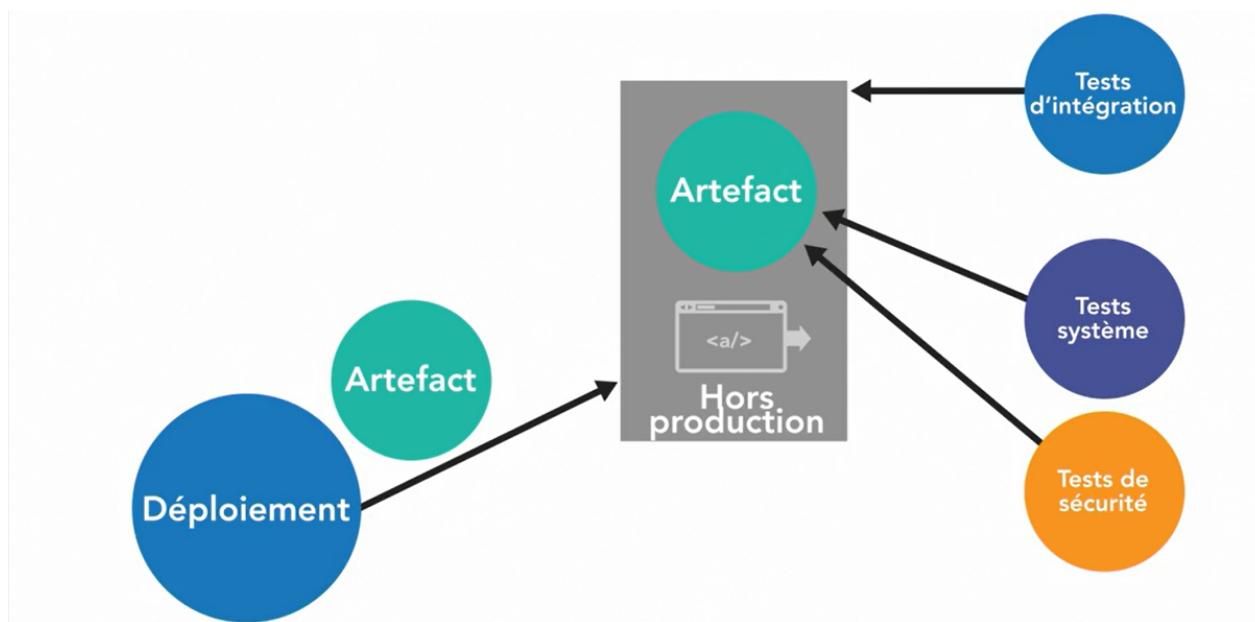
Livraison Continue

- Nombreuses pièces mobiles
- Amélioration des chances de réussite

CI/CD

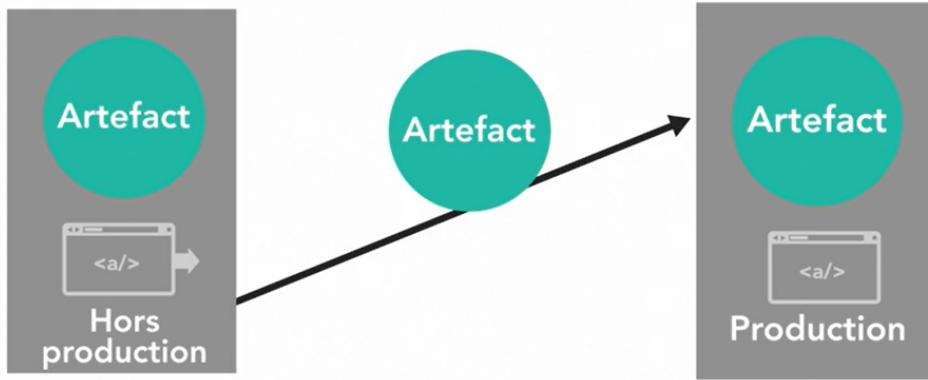


CD Hors Production



CD Production

Une fois l'artefact valider, il sera déployer automatiquement



La livraison continue est une exigence pour atteindre l'agilité dans une architecture de microservices.

Connaître Quelques Architectures Hybrides Viables

Architecture Hybride De Service

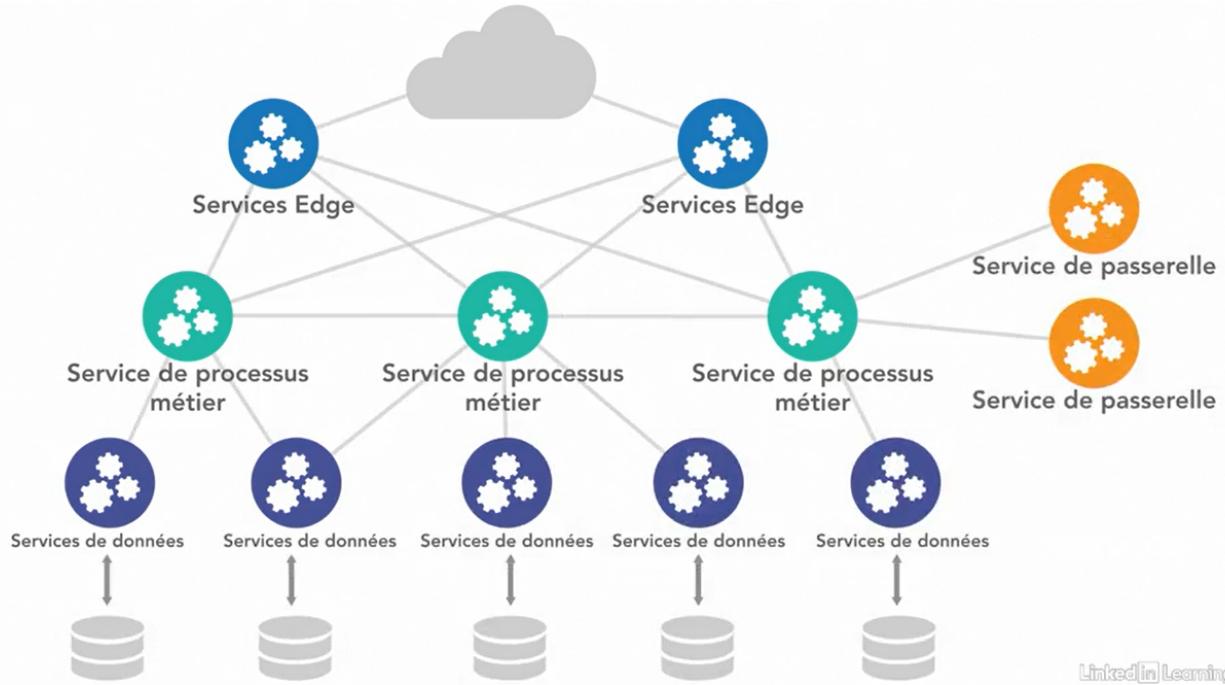
- De nombreux leaders s'opposent à ce modèle
- Il évite les dépendances circulaires
- Modélise une architecture n-tier via des services plutôt que des modules

Architecture N-tiers Hybrides

Dans le modèle hiérarchique n-tiers, nous pouvons définir différentes classes de services:

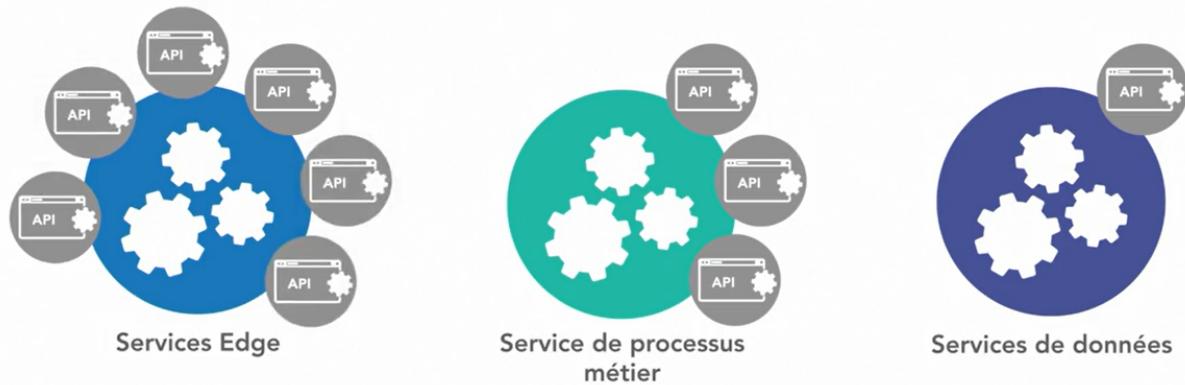
- le service de données est une classe commune qui expose la logique des données spécifique au domaine au monde extérieur
- le service des processus métier constitue une autre classe commune qui définit précisément des processus métiers de haut niveau
- nous pouvons également définir des services de passerelles qui créent des abstractions vers des dépendances externes
- et enfin nous pouvons définir des services Edge qui exposent nos données et nos processus métiers aux monde extérieurs
- une fois que nous avons une taxonomie claire, nous pouvons définir des règles sur ce que les services peuvent consommer, étudier les risques liés aux appels circulaires et interdire qu'un service appelle aucun autre service de donnée sans être impliqué dans un processus métier. La consommation de service de passerelle intervient aussi via les mêmes

processus métier et non entre eux.



Architecture De Services

- Base de données sous-jacente unique
- Exploitation des services pour gérer la décomposition
- Avantages en termes d'agilité sans modification des banques



Parcourir Les Considérations Relatives A La Conception

Pour concevoir une architecture microservice voici quelques points essentiels à tenir compte:

L'intégration continue et la livraison continue sont le premier aspect à prendre en compte pour votre conception.

La journalisation et le traçage sont le second aspect à prendre en compte pour votre conception.

- Concernant le code nous devons envisager de tirer parti de la conception piloter le domaine
- nous devons effectuer une véritable analyse de l'ensemble du système et utiliser ces connaissances pour développer nos services
- nous devons réfléchir à la création de nos services et à la fonction qu'ils vont exécuter
- nous devons également penser si nous allons exploiter des services de données dédier ou intégrer les données dans les processus métiers
- devrions-nous intégrer les services ensembles pour obtenir les transactions avec leur propriétés ACID là où cela est requis ou devrions-nous élaborer des stratégie basée sur la cohérence éventuelle des données

Conception Des Mécanismes Pour Contrôler La Latence

- Évaluer si possible l'utilisation de code non bloquant
- Standardiser votre pile

Beaucoup de développeur n'aime entendre la standardisation, mais les équipes qui réussissent le mieux avec les architectures de microservices ont opté pour la standardisation.

La standardisation permet de déplacer les ressources plus facilement en fonction de l'évolution des besoins métiers, de plus au départ concevons nos systèmes de manière asynchrone. Essayons de créer chaque service avec des opérations asynchrones jusqu'à ce qu'il soit approuvé que nous en avons besoins des opérations synchrones.

Discerner Les Compromis

Payer La <<Distribution Tax >> Offre Plusieurs Avantages

- Capacité de distribution
- Délimitation claire des modules
- Capacité de mise à l'échelle

Prenons le cas d'amazone qui opère à l'échelle mondiale et qui clairement atteint une pic d'activité à la veille de la fête de paque il serait quasiment impossible pour elle de gérer son système dans le cadre d'un déploiement monolithique. Elle gère la **distribution tax** modèle hautement disponible qui constitue un avantage évident pour opérer à l'échelle mondiale.

Problèmes Liés A La Complexité

- Capacité de mise à l'échelle et déploiements
- Grand nombre de pièces mobiles

Pratiques De Développement Polyglotte

- Nombreux avantages et inconvénients
- S'en servir comme d'un outil
- Envisager la standardisation

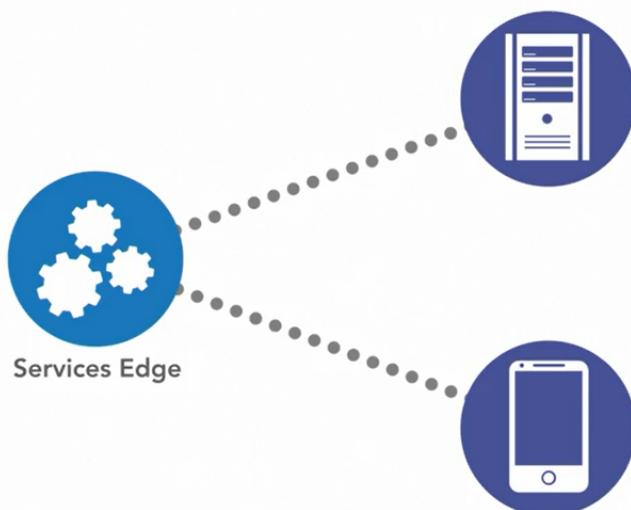
Le développement polyglotte s'accompagne avec une complication de la gestion des service en production.

Voir Les Avantages Des Services De Pointe

La couche d'API ne doit être utilisée que comme couche proxy.

Services Edge Sortants

- Servent à exposer les besoins spécifiques au client au monde extérieur



Services Edge Entrantes / De Traduction

- Vous extraient des dépendances tierces



Avantages D'un service Edge

- Permet de gérer les transformations du code comme pour le reste du code du système
- Fournit une interface cohérente même si les services sous-jacents changent

Adopter Une Culture DevOps

Ne jamais sous-estimer la culture.



Surveiller un système de microservices pour s'assurer que le retard n'ait pas d'impact majeur.

Automatisation Des Tâches



Version de codes Et Gestion Automatique



Notions Aborder Dans cette Formation

Histoire des microservices

Concepts clés des microservices

Concepts avancés des microservices

Architecture et culture