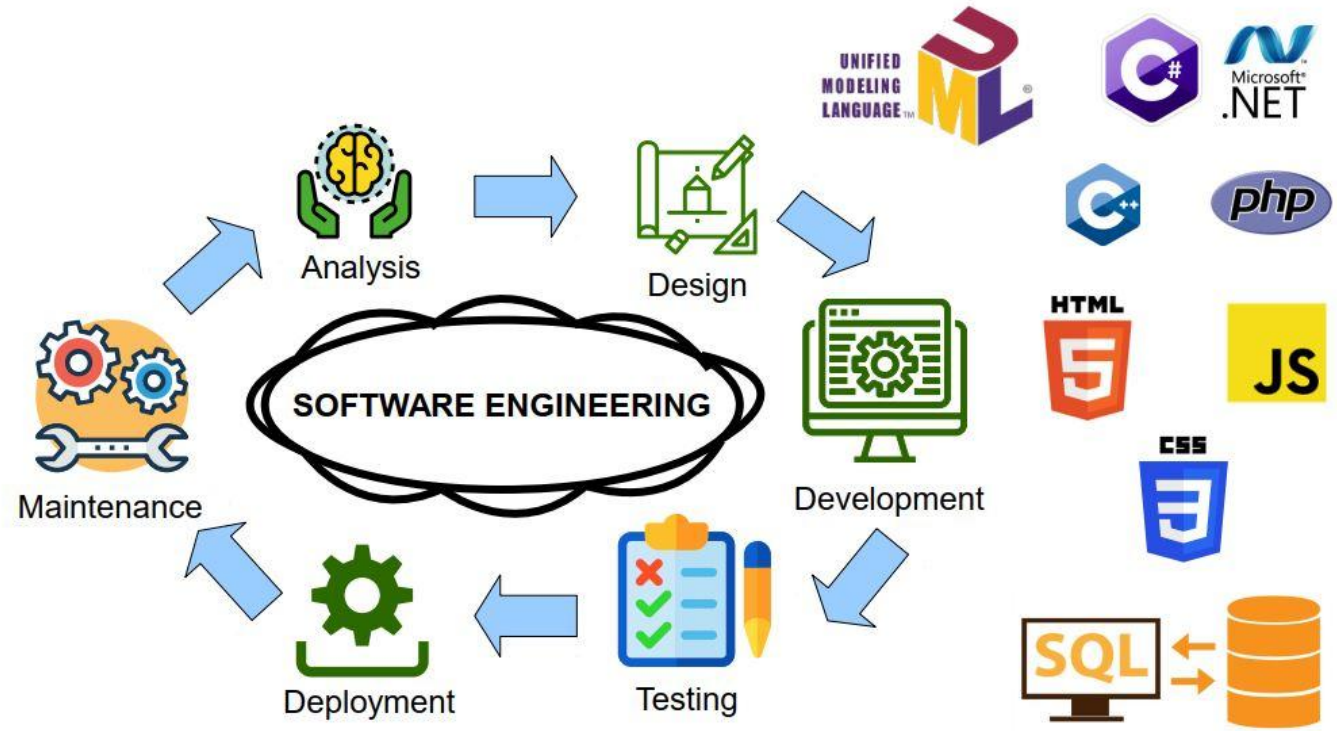
Architecture Logicielle



Dr El Hadji Bassirou TOURE

Université Cheikh Anta Diop de Dakar

2024 - 2025

Architecture microservices

Services logiciels

- Un service logiciel est un composant logiciel auquel on peut accéder à partir d'ordinateurs distants via l'internet. À partir d'une entrée, un service produit une sortie correspondante, sans effets secondaires.
  - On accède au service par son interface publiée et tous les détails de l'implémentation du service sont cachés.
  - Les services ne conservent aucun état interne. Les informations relatives à l'état sont soit stockées dans une base de données, soit conservées par le demandeur du service.
- Lorsqu'une demande de service est formulée, les informations sur l'état peuvent être incluses dans la demande et les informations actualisées sur l'état sont renvoyées dans le cadre du résultat du service.
- Comme il n'y a pas d'état local, les services peuvent être réaffectés dynamiquement d'un serveur virtuel à un autre et répliqués sur plusieurs serveurs.

Les services web modernes

- Après diverses expériences menées dans les années 1990 dans le domaine de l'informatique orientée services, l'idée de "grands" services web est apparue au début des années 2000.
- Ceux-ci étaient basés sur des protocoles et des normes XML tels que SOAP pour l'interaction des services et WSDL pour la description des interfaces.
- La plupart des services logiciels n'ont pas besoin de la généralité inhérente à la conception des protocoles de services web.
- Par conséquent, les systèmes modernes orientés services utilisent des protocoles d'interaction de services plus simples et plus légers, dont les frais généraux sont moindres et dont l'exécution est par conséquent plus rapide.

Microservices

- Les microservices sont des services à petite échelle, sans état, qui ont une responsabilité unique. Ils sont combinés pour créer des applications.
- Ils sont totalement indépendants et possèdent leur propre base de données et leur propre code de gestion de l'interface utilisateur.
- Les produits logiciels qui utilisent des microservices ont une architecture microservices.
- Si vous devez créer des produits logiciels basés sur le cloud qui soient adaptables, évolutifs et résilients, je vous recommande de les concevoir autour d'une architecture de microservices.

Un exemple de microservice

Authentification du système

- Enregistrement de l'utilisateur, où les utilisateurs fournissent des informations sur leur identité, leurs informations de sécurité, leur numéro de téléphone mobile (portable) et leur adresse électronique.
- Authentification par identifiant/mot de passe.
- Authentification à deux facteurs à l'aide d'un code envoyé au téléphone portable.
- Gestion des informations sur l'utilisateur, par exemple changement de mot de passe ou de numéro de téléphone portable.
- Réinitialisation d'un mot de passe oublié.

Chacune de ces fonctions pourrait être mise en œuvre sous la forme d'un service distinct utilisant une base de données centrale partagée pour contenir les informations d'authentification.

Cependant, ces fonctionnalités sont trop importantes pour être des microservices. Pour identifier les microservices susceptibles d'être utilisés dans le système d'authentification, vous devez décomposer les fonctionnalités à gros grain en fonctions plus détaillées.

Ventilation fonctionnelle des fonctions d'authentification

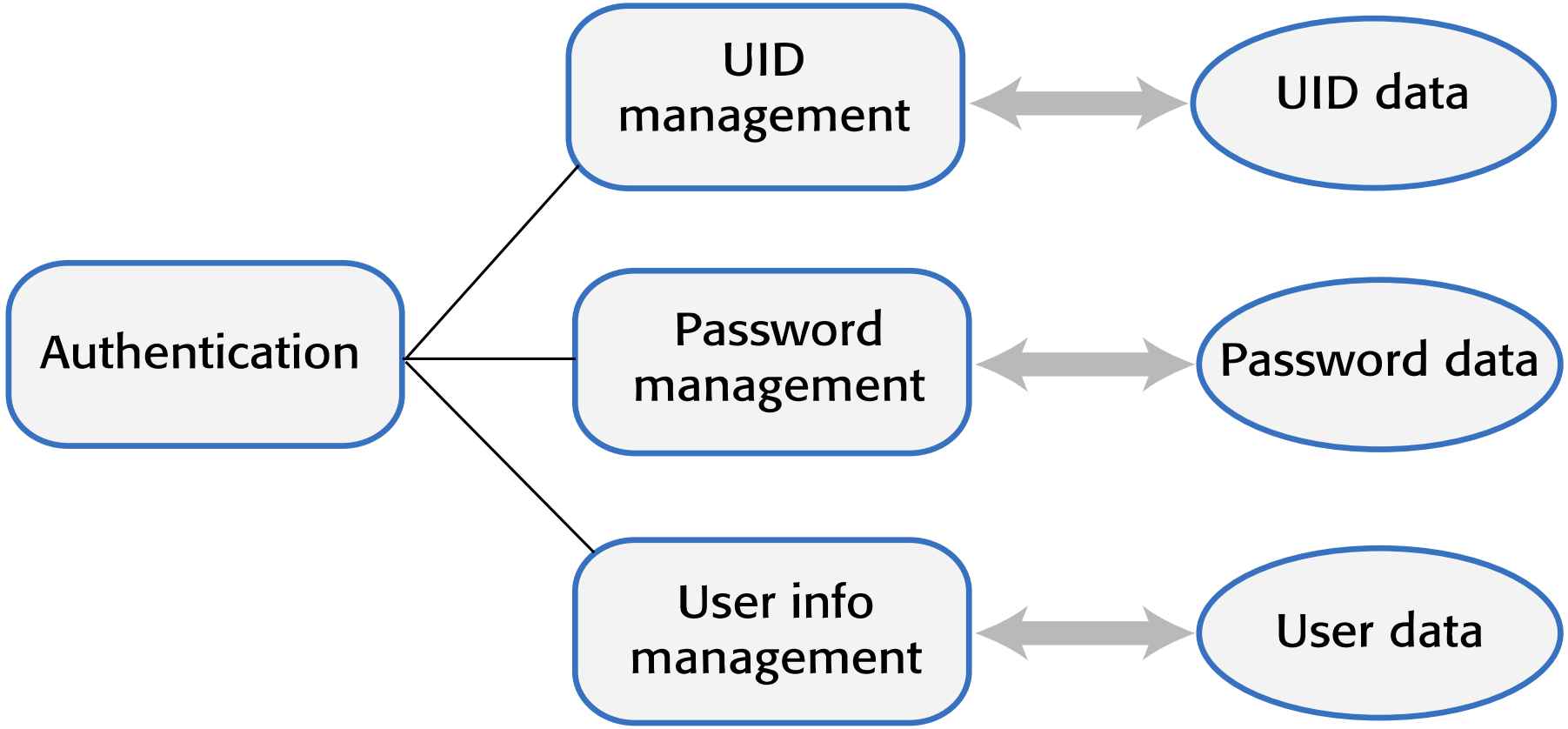
User registration

Setup new login id
Setup new password
Setup password recovery information
Setup two-factor authentication
Confirm registration

Authenticate using UID/password

Get login id
Get password
Check credentials
Confirm authentication

Microservices d'authentification



Caractéristiques des microservices

- Autonome
  Les microservices n'ont pas de dépendances externes. Ils gèrent leurs propres données et mettent en œuvre leur propre interface utilisateur.
- Les microservices légers communiquent à l'aide de protocoles légers, de sorte que les frais généraux de communication des services sont faibles.
- Indépendant de la mise en œuvre Les microservices peuvent être mis en œuvre à l'aide de différents langages de programmation et peuvent utiliser différentes technologies (par exemple, différents types de bases de données) dans leur mise en œuvre.
- Déploiement indépendant Chaque microservice s'exécute dans son propre processus et peut être déployé indépendamment, à l'aide de systèmes automatisés.
- Les microservices doivent mettre en œuvre les capacités et les besoins de l'entreprise, plutôt que de simplement fournir un service technique.

Communication entre microservices

- Les microservices communiquent en échangeant des messages.
- Un message envoyé entre services comprend des informations administratives, une demande de service et les données nécessaires pour fournir le service demandé.
- Les services renvoient une réponse aux messages de demande de service.
  - Un service d'authentification peut envoyer un message à un service de connexion qui comprend le nom saisi par l'utilisateur.
  - La réponse peut être un jeton associé à un nom d'utilisateur valide ou une erreur indiquant qu'il n'y a pas d'utilisateur enregistré.

Caractéristiques des microservices

- Un microservice bien conçu doit présenter une forte cohésion et un faible couplage.
  - La cohésion est une mesure du nombre de relations que les parties d'un composant entretiennent entre elles. Une cohésion élevée signifie que toutes les parties nécessaires à la réalisation des fonctionnalités du composant sont incluses dans ce dernier.
  - Le couplage est une mesure du nombre de relations qu'un composant entretient avec d'autres composants du système. Un couplage faible signifie que les composants n'ont pas beaucoup de relations avec d'autres composants.
- Chaque microservice doit avoir une responsabilité unique, c'est-à-dire qu'il ne doit faire qu'une seule chose et la faire bien.
  - Cependant, il est difficile de définir "une seule chose" d'une manière qui soit applicable à tous les services.
  - La responsabilité ne signifie pas toujours une activité unique et fonctionnelle.

Fonctionnalité de gestion des mots de passe

User functions

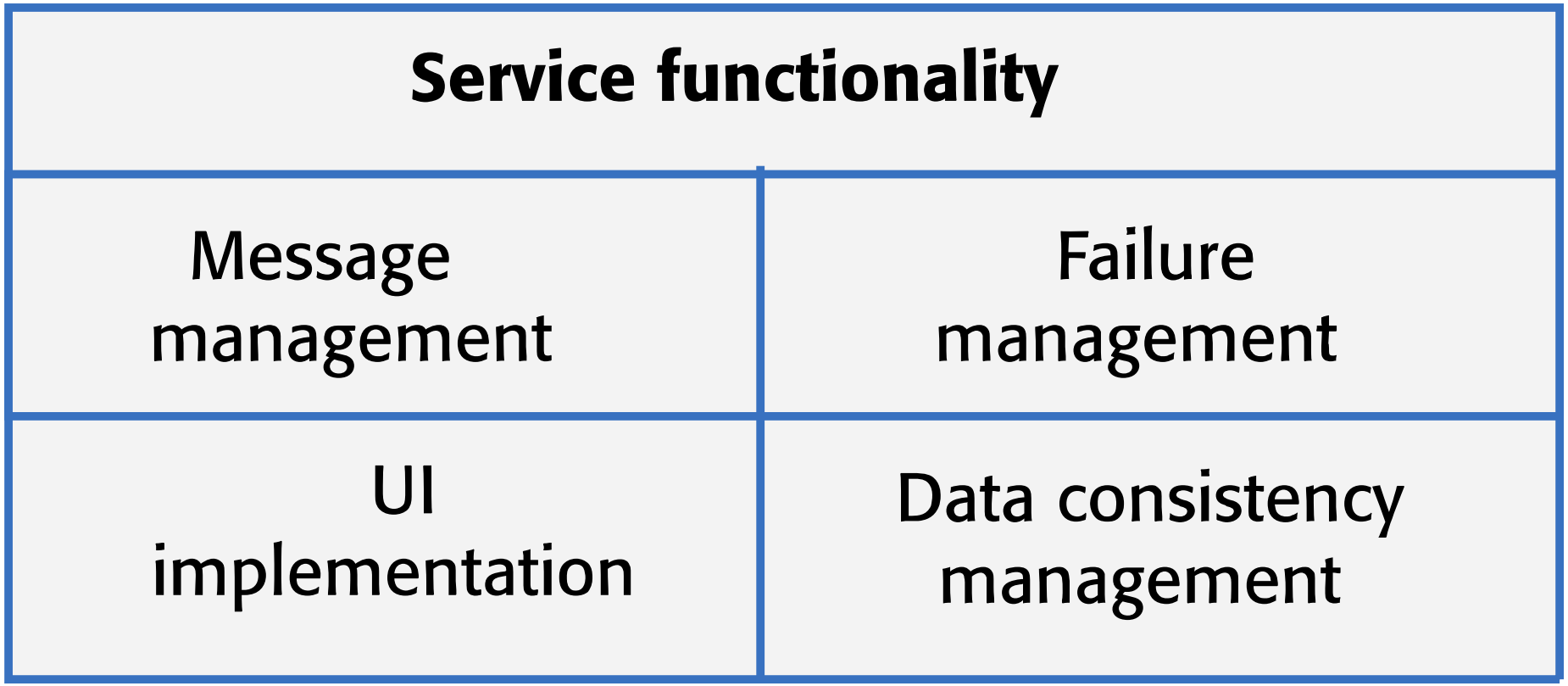
- Create password
- Change password
- Check password
- Recover password

Supporting functions

- Check password validity
- Delete password
- Backup password database
- Recover password database
- Check database integrity
- Repair password DB

Code de support des microservices

Microservice X



Architecture microservices

- Une architecture de microservices est un style architectural - une manière éprouvée de mettre en œuvre une architecture logicielle logique.
- Ce style architectural répond à deux problèmes posés par les applications monolithiques
  - L'ensemble du système doit être reconstruit, retesté et redéployé en cas de changement. Ce processus peut être lent, car les modifications apportées à une partie du système peuvent avoir des répercussions négatives sur d'autres composants.
  - Au fur et à mesure que la demande augmente, l'ensemble du système doit être mis à l'échelle, même si la demande est localisée à un petit nombre de composants du système qui mettent en œuvre les fonctions les plus populaires du système.

Avantages de l'architecture microservices

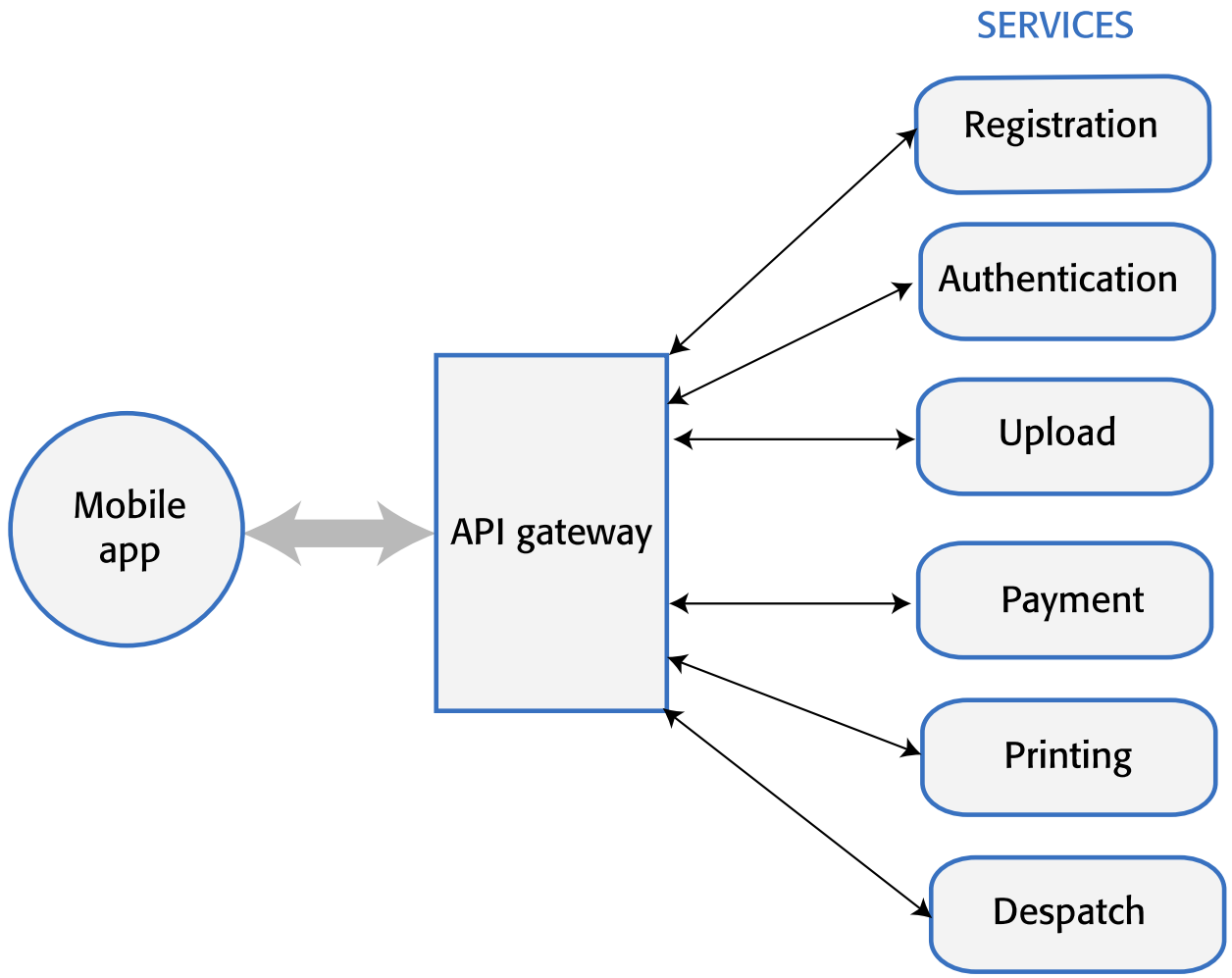
- Les microservices sont autonomes et s'exécutent dans des processus distincts.
- Dans les systèmes basés sur le cloud, chaque microservice peut être déployé dans son propre conteneur. Cela signifie qu'un microservice peut être arrêté et redémarré sans affecter les autres parties du système.
- Si la demande d'un service augmente, des répliques du service peuvent être rapidement créées et déployées. Ces répliques ne nécessitent pas de serveur plus puissant, de sorte que l'extension est généralement beaucoup moins coûteuse que l'augmentation.

Système d'impression de photos pour appareils mobiles

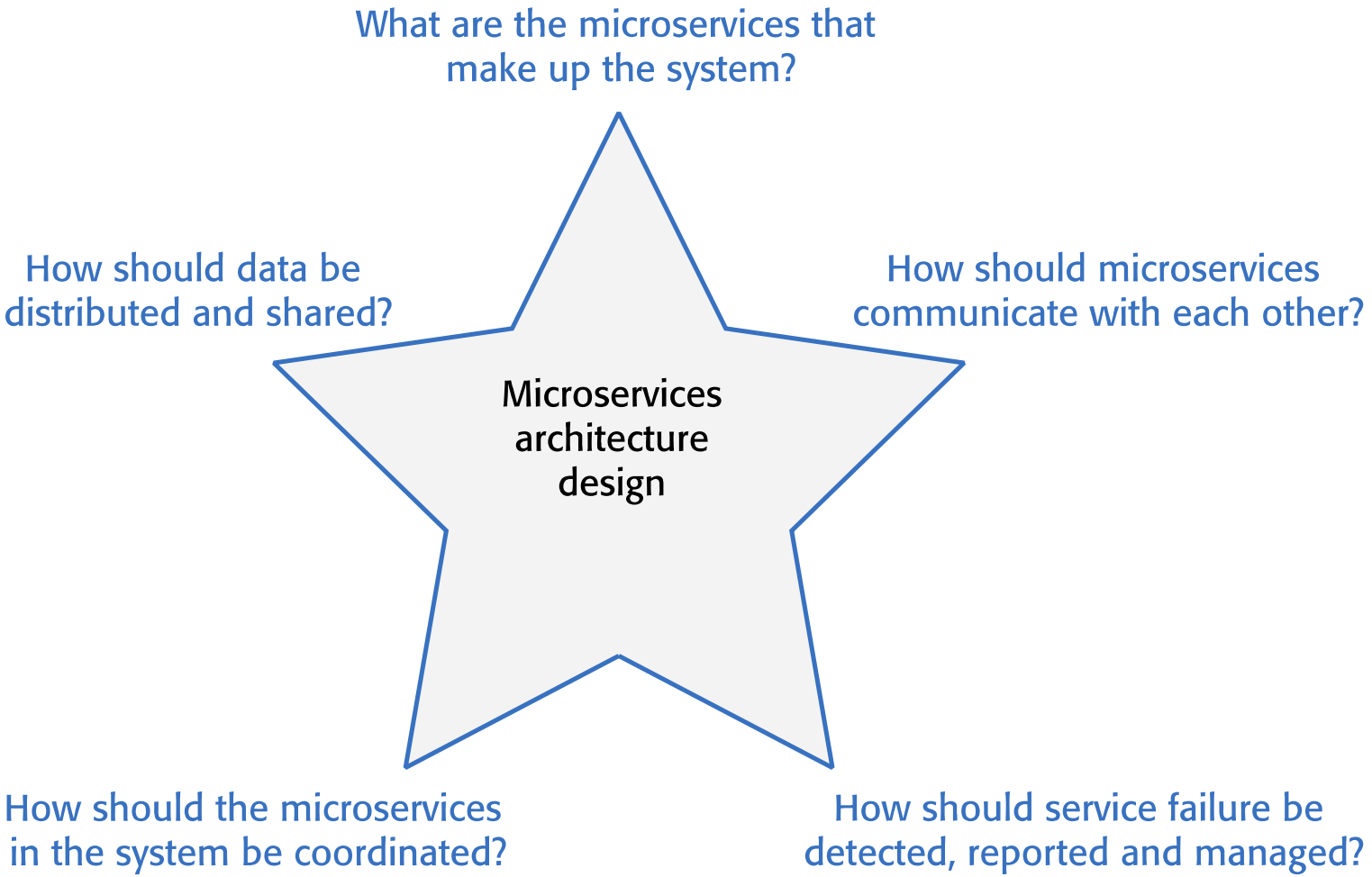
- Imaginez que vous développiez un service d'impression de photos pour les appareils mobiles. Les utilisateurs peuvent télécharger des photos sur votre serveur depuis leur téléphone ou spécifier des photos de leur compte Instagram qu'ils souhaitent voir imprimées. Les impressions peuvent être réalisées à différentes tailles et sur différents supports.

- Les utilisateurs peuvent choisir la taille et le support d'impression. Par exemple, ils peuvent décider d'imprimer une photo sur un mug ou un T-shirt. Les tirages ou autres supports sont préparés puis envoyés à leur domicile. Ils paient les tirages soit en utilisant un service de paiement tel qu'Android ou Apple Pay, soit en enregistrant une carte de crédit auprès du fournisseur de services d'impression.

Architecture de microservices pour un système d'impression de photos



Architecture de microservices - questions clés de conception



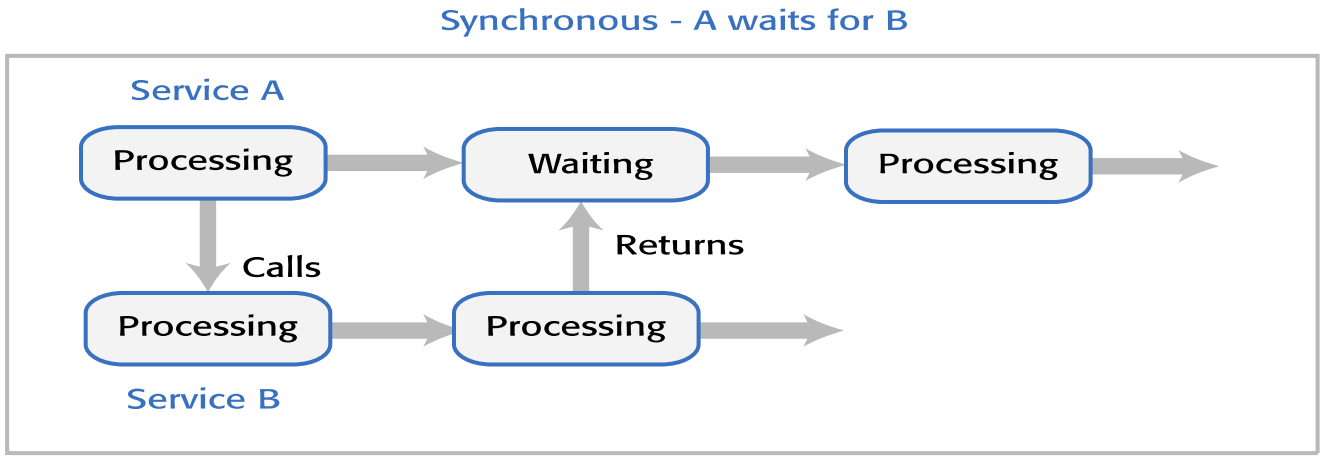
Lignes directrices pour la décomposition

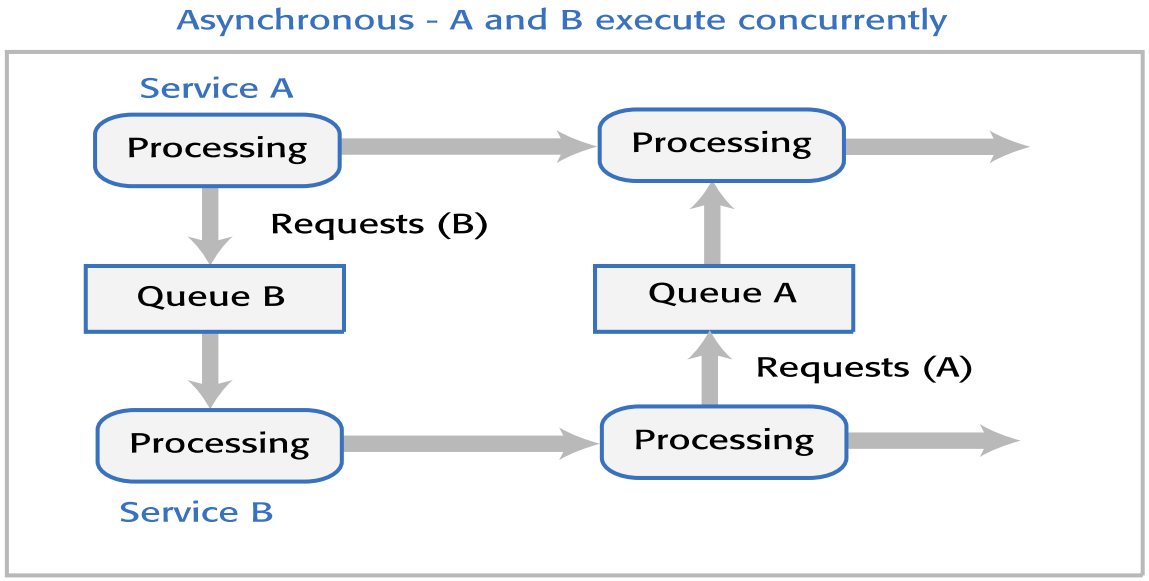
- Équilibrer la fonctionnalité à grain fin et la performance du système
  - Les services à fonction unique signifient que les changements sont limités à un nombre réduit de services, mais qu'ils nécessitent des communications de service pour mettre en œuvre la fonctionnalité de l'utilisateur. Cela ralentit un système en raison de la nécessité pour chaque service de regrouper et de dégrouper les messages envoyés par d'autres services.
- Suivre le "principe de fermeture commune
  - Les éléments d'un système susceptibles d'être modifiés en même temps doivent être situés dans le même service. La plupart des exigences nouvelles et modifiées ne devraient donc concerner qu'un seul service.
- Associer les services aux capacités de l'entreprise
  - Une capacité métier est un domaine discret de fonctionnalité métier qui relève de la responsabilité d'une personne ou d'un groupe. Vous devez identifier les services nécessaires pour soutenir chaque capacité métier.
- Concevoir les services de manière à ce qu'ils n'aient accès qu'aux données dont ils ont besoin
  - S'il existe un chevauchement entre les données utilisées par différents services, vous devez disposer d'un mécanisme permettant de propager les modifications de données à tous les services utilisant les mêmes données.

Communications entre services

- Les services communiquent en échangeant des messages qui contiennent des informations sur l'expéditeur du message, ainsi que les données qui constituent l'entrée ou la sortie de la requête.
- Lorsque vous concevez une architecture de microservices, vous devez établir une norme de communication que tous les microservices doivent respecter. Voici quelques-unes des décisions clés que vous devez prendre
  - L'interaction entre les services doit-elle être synchrone ou asynchrone ?
  - Les services doivent-ils communiquer directement ou par l'intermédiaire d'un logiciel intermédiaire de courtage de messages ?
  - quel protocole doit être utilisé pour les messages échangés entre les services ?

Interaction entre microservices synchrones et asynchrones



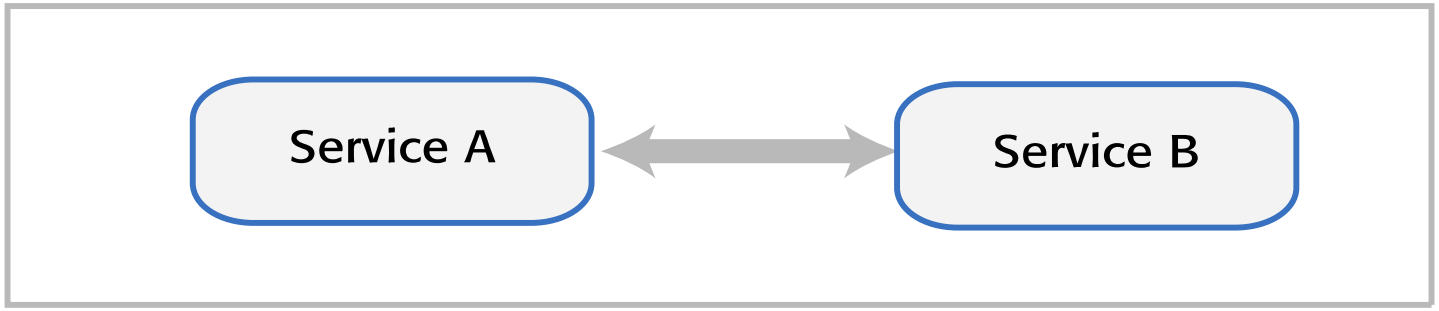


Interaction synchrone et asynchrone

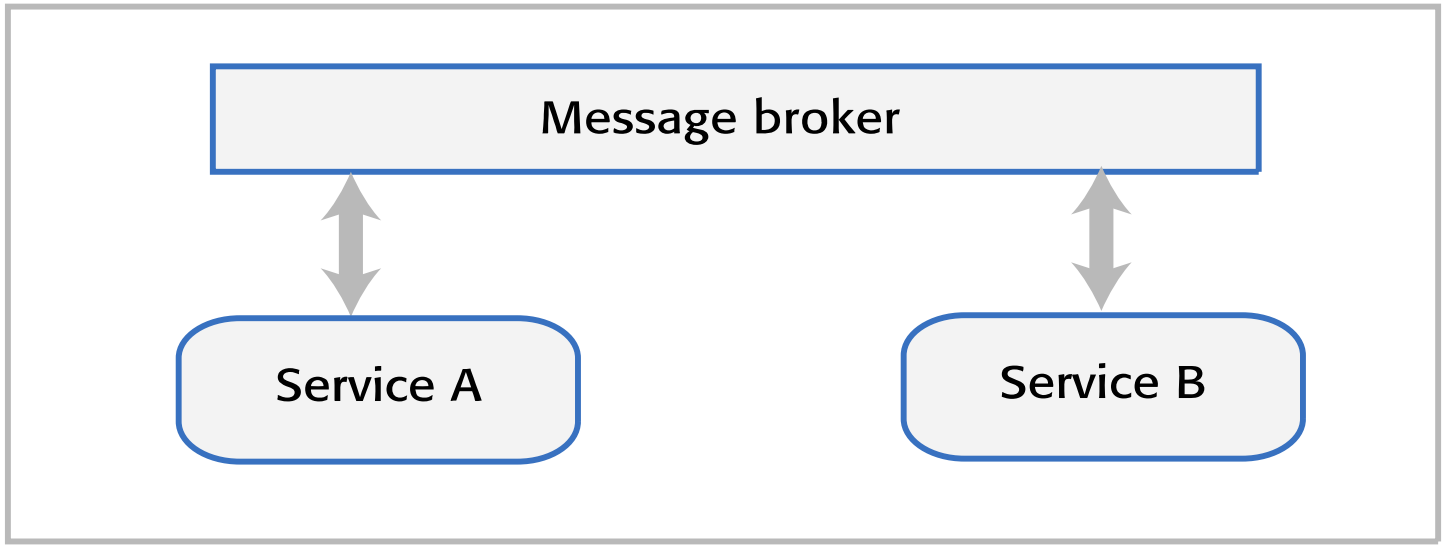
- Dans une interaction synchrone, le service A émet une requête au service B. Le service A suspend alors son traitement pendant que B traite la requête.
- Il attend que le service B ait renvoyé les informations requises avant de poursuivre l'exécution.
- Dans une interaction asynchrone, le service A émet la requête qui est mise en file d'attente pour être traitée par le service B. A poursuit alors le traitement sans attendre que B ait terminé ses calculs.
- Quelque temps plus tard, le service B complète la requête précédente du service A et met le résultat en file d'attente pour qu'il soit récupéré par A.
- Le service A doit donc vérifier périodiquement sa file d'attente pour voir si un résultat est disponible.

Communication directe et indirecte entre services

Direct communication - A and B send messages to each other



Indirect communication - A and B communicate through a message broker



Communication directe et indirecte de services

- La communication directe entre services nécessite que les services qui interagissent connaissent l'adresse de chacun d'entre eux.
- Les services interagissent en envoyant des requêtes directement à ces adresses.
- La communication indirecte implique de nommer le service requis et d'envoyer cette demande à un courtier de messages (parfois appelé bus de messages).
- Le courtier de messages est alors chargé de trouver le service qui peut répondre à la demande.

Conception des données des microservices

- Vous devez isoler les données au sein de chaque service système en limitant autant que possible le partage des données.
- Si le partage des données est inévitable, vous devez concevoir les microservices de manière à ce que la plupart des échanges se fassent en "lecture seule", avec un nombre minimal de services responsables des mises à jour des données.
- Si les services sont répliqués dans votre système, vous devez inclure un mécanisme capable de maintenir la cohérence des copies de la base de données utilisées par les services répliqués.

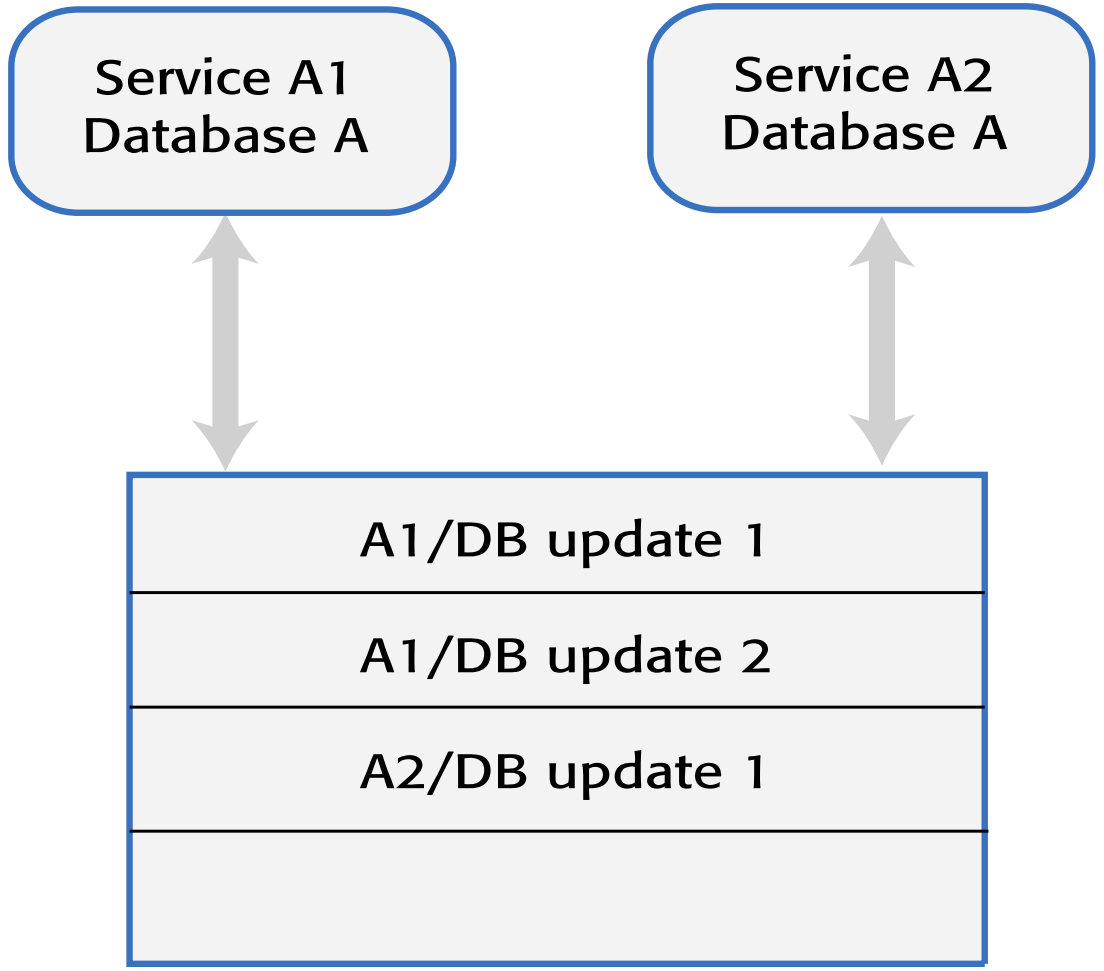
Gestion des incohérences

- Une transaction ACID regroupe un ensemble de mises à jour de données en une seule unité, de sorte que soit toutes les mises à jour sont terminées, soit aucune ne l'est. Les transactions ACID ne sont pas pratiques dans une architecture microservices.
- Les bases de données utilisées par différents microservices ou répliques de microservices ne doivent pas nécessairement être totalement cohérentes en permanence.
- Incohérence des données dépendantes
  - Les actions ou les défaillances d'un service peuvent entraîner l'incohérence des données gérées par un autre service.
- Incohérence des répliques
  - Plusieurs répliques du même service sont exécutées simultanément. Chacune d'entre elles possède sa propre copie de la base de données et met à jour sa propre copie des données du service. Vous devez trouver un moyen de rendre ces bases de données "éventuellement cohérentes" afin que toutes les répliques travaillent sur les mêmes données.

Cohérence éventuelle

- La cohérence éventuelle est une situation dans laquelle le système garantit que les bases de données finiront par être cohérentes.
- Vous pouvez mettre en œuvre la cohérence éventuelle en gérant un journal des transactions.
- Lorsqu'une modification est apportée à la base de données, elle est enregistrée dans un journal des "mises à jour en attente".
- Les autres instances de service consultent ce journal, mettent à jour leur propre base de données et indiquent qu'elles ont effectué la modification.

Utilisation d'un journal des transactions en attente

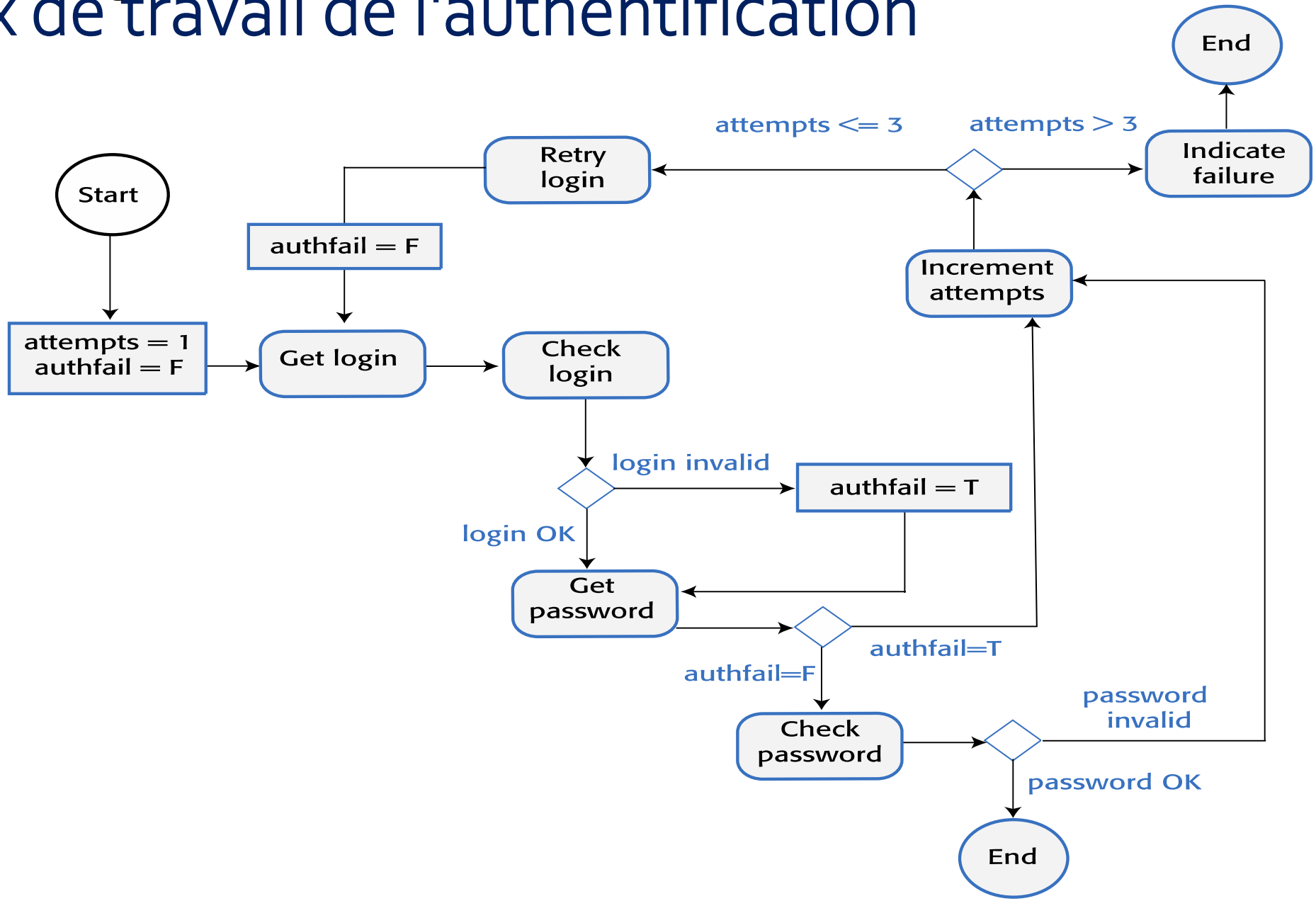


Pending transactions log

Coordination des services

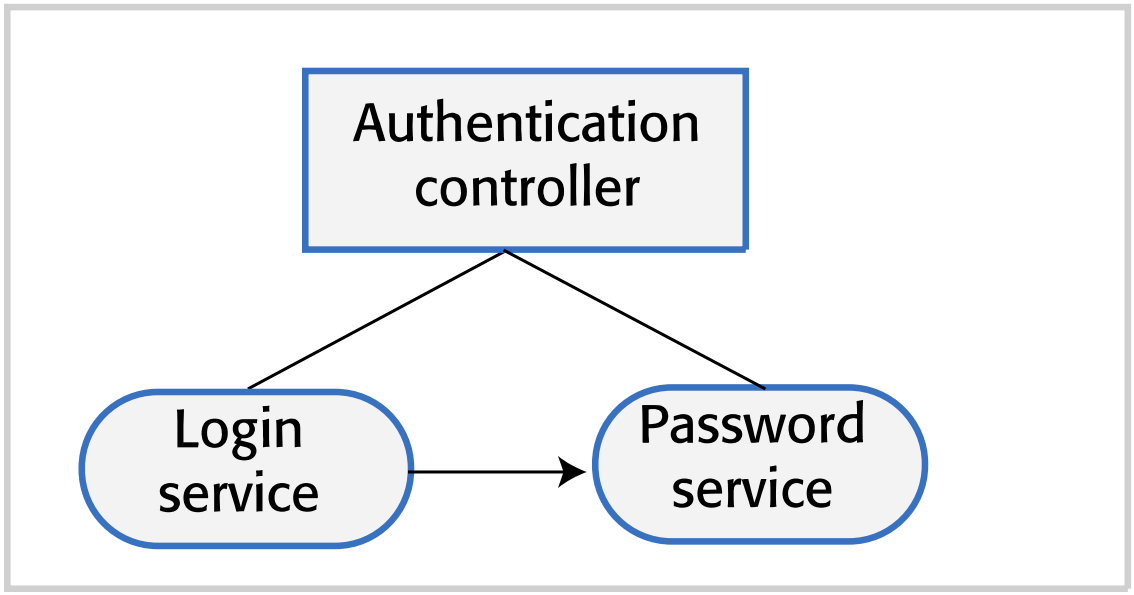
- La plupart des sessions d'utilisateurs impliquent une série d'interactions dans lesquelles les opérations doivent être effectuées dans un ordre spécifique.
- C'est ce qu'on appelle un flux de travail.
  - Un flux de travail pour l'authentification par UID/mot de passe montre les étapes de l'authentification d'un utilisateur.
  - Dans cet exemple, l'utilisateur a droit à trois tentatives de connexion avant que le système n'indique que la connexion a échoué.

Flux de travail de l'authentification

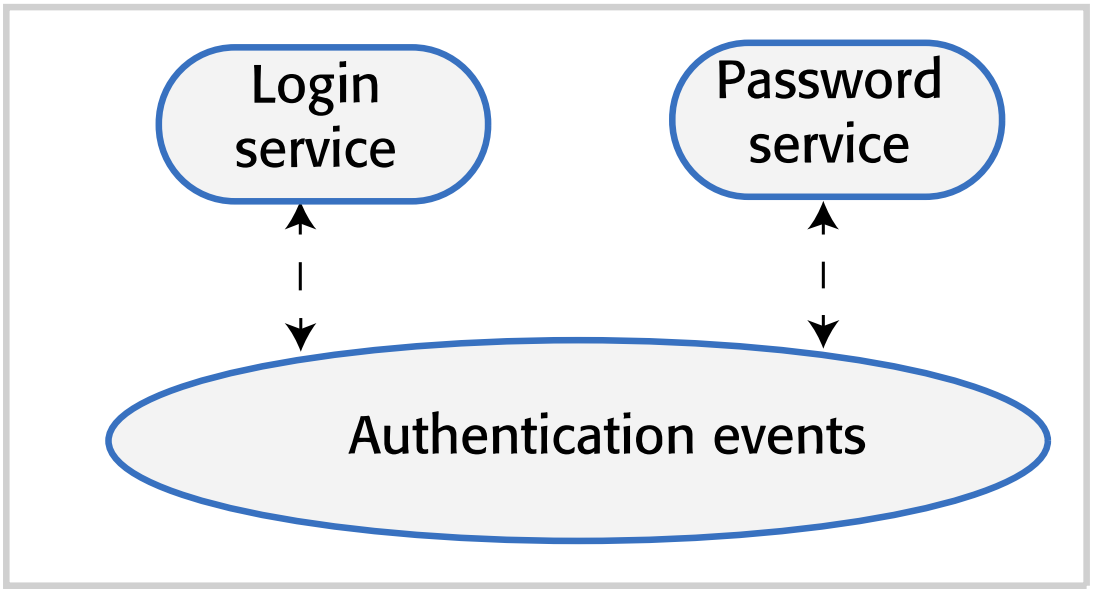


Orchestration et chorégraphie

Service orchestration



Service choreography



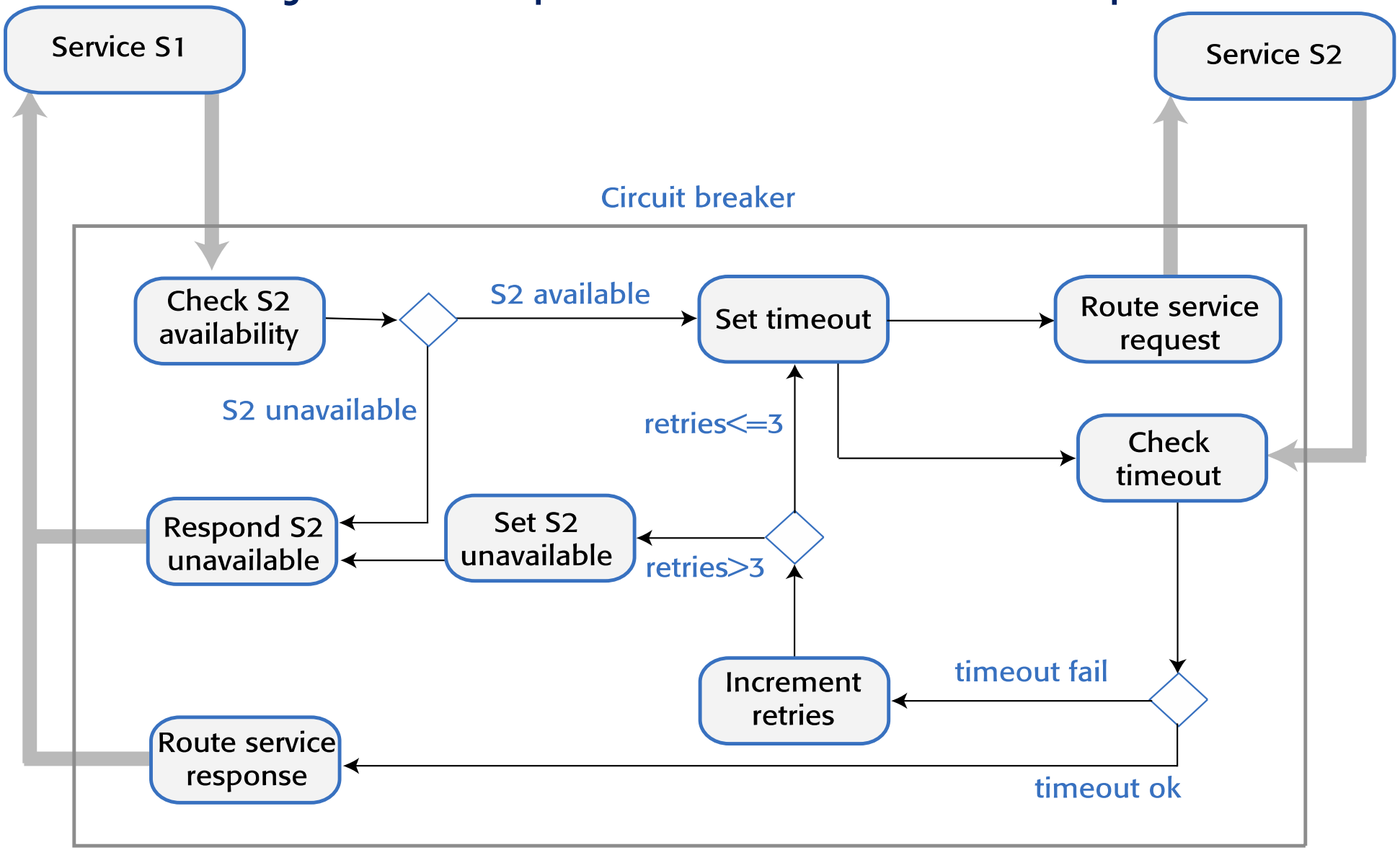
Types de défaillances dans un système de microservices

- Défaillance interne du service Il s'agit de conditions détectées par le service et qui peuvent être signalées au client du service dans un message d'erreur. Un exemple de ce type de défaillance est un service qui prend un URL en entrée et découvre qu'il s'agit d'un lien non valide.Ceux-ci étaient basés sur des protocoles et des normes XML tels que SOAP pour l'interaction des services et WSDL pour la description des interfaces.
- Défaillance interne du service Il s'agit de conditions détectées par le service et qui peuvent être signalées au client du service dans un message d'erreur. Un exemple de ce type de défaillance est un service qui prend un URL en entrée et découvre qu'il s'agit d'un lien non valide.
- Défaillance des performances du service Les performances du service se dégradent jusqu'à atteindre un niveau inacceptable. Cela peut être dû à une charge importante ou à un problème interne au service. La surveillance externe des services peut être utilisée pour détecter les défaillances de performance et les services qui ne répondent pas.

Délais d'attente et disjoncteurs

- Un délai d'attente est un compteur associé aux demandes de service et qui commence à fonctionner lorsque la demande est formulée.
- Lorsque le compteur atteint une valeur prédéfinie, par exemple 10 secondes, le service appelant considère que la demande de service a échoué et agit en conséquence.
- Le problème de cette approche est que chaque appel à un "service en échec" est retardé par la valeur du délai, ce qui ralentit l'ensemble du système.
- Au lieu d'utiliser explicitement des délais d'attente lorsqu'un appel de service est effectué, il suggère d'utiliser un disjoncteur. À l'instar d'un disjoncteur électrique, celui-ci refuse immédiatement l'accès à un service défaillant sans les délais associés aux temporisations.

Utilisation d'un disjoncteur pour faire face à une panne de service



Services RESTful

- Le style architectural REST (REpresentational State Transfer) repose sur l'idée du transfert de représentations de ressources numériques d'un serveur à un client.
  - On peut considérer qu'une ressource est un ensemble de données telles que les détails d'une carte de crédit, le dossier médical d'un individu, un magazine ou un journal, le catalogue d'une bibliothèque, etc.
  - On accède aux ressources par leur URI unique et les services RESTful opèrent sur ces ressources.
- Il s'agit de l'approche fondamentale utilisée sur le web, où la ressource est une page à afficher dans le navigateur de l'utilisateur.
  - Une représentation HTML est générée par le serveur en réponse à une requête HTTP GET et est transférée au client pour être affichée par un navigateur ou une application spécialisée.

Principes des services RESTful

- Utiliser les verbes HTTP Les méthodes de base définies dans le protocole HTTP (GET, PUT, POST, DELETE) doivent être utilisées pour accéder aux opérations mises à disposition par le service.
- Services sans état Les services ne doivent jamais maintenir d'état interne. Comme je l'ai déjà expliqué, les microservices sont sans état et s'inscrivent donc dans ce principe.
- Adressable par URI Toutes les ressources doivent avoir un URI, avec une structure hiérarchique, qui est utilisé pour accéder aux sous-ressources.
- Utiliser XML ou JSON Les ressources doivent normalement être représentées en JSON ou XML, ou les deux. D'autres représentations, telles que des représentations audio et vidéo, peuvent être utilisées le cas échéant.

Opérations de service RESTful

- Create Mise en œuvre à l'aide de HTTP POST, qui crée la ressource avec l'URI donné. Si la ressource a déjà été créée, une erreur est renvoyée.
- Read Mise en œuvre à l'aide de HTTP GET, qui lit la ressource et renvoie sa valeur. Les opérations GET ne doivent jamais mettre à jour une ressource, de sorte que des opérations GET successives sans opérations PUT intermédiaires renvoient toujours la même valeur.
- Mise à jour Mise en œuvre à l'aide de HTTP PUT, qui modifie une ressource existante. PUT ne doit pas être utilisé pour la création de ressources.
- Delete Mise en œuvre à l'aide de l'opération HTTP DELETE, qui rend la ressource inaccessible à l'aide de l'URI spécifié. La ressource peut être physiquement supprimée ou non.

Système d'information routière

- Imaginons un système qui conserve des informations sur les incidents, tels que les retards de circulation, les travaux routiers et les accidents sur un réseau routier national. Ce système est accessible via un navigateur à l'aide de l'URL :
  - https://trafficinfo.net/incidents/
- Les utilisateurs peuvent interroger le système pour découvrir les incidents sur les routes qu'ils prévoient d'emprunter.
- Lors de la mise en œuvre d'un service web RESTful, vous devez concevoir la structure des ressources de manière à ce que les incidents soient organisés de manière hiérarchique.
  - Par exemple, les incidents peuvent être enregistrés en fonction de l'identifiant de la route (par exemple A90), de la localisation (par exemple stonehaven), de la direction de la chaussée (par exemple nord) et d'un numéro d'incident (par exemple 1). Il est donc possible d'accéder à chaque incident à l'aide de son URI :
  - https://trafficinfo.net/incidents/A90/stonehaven/north/1

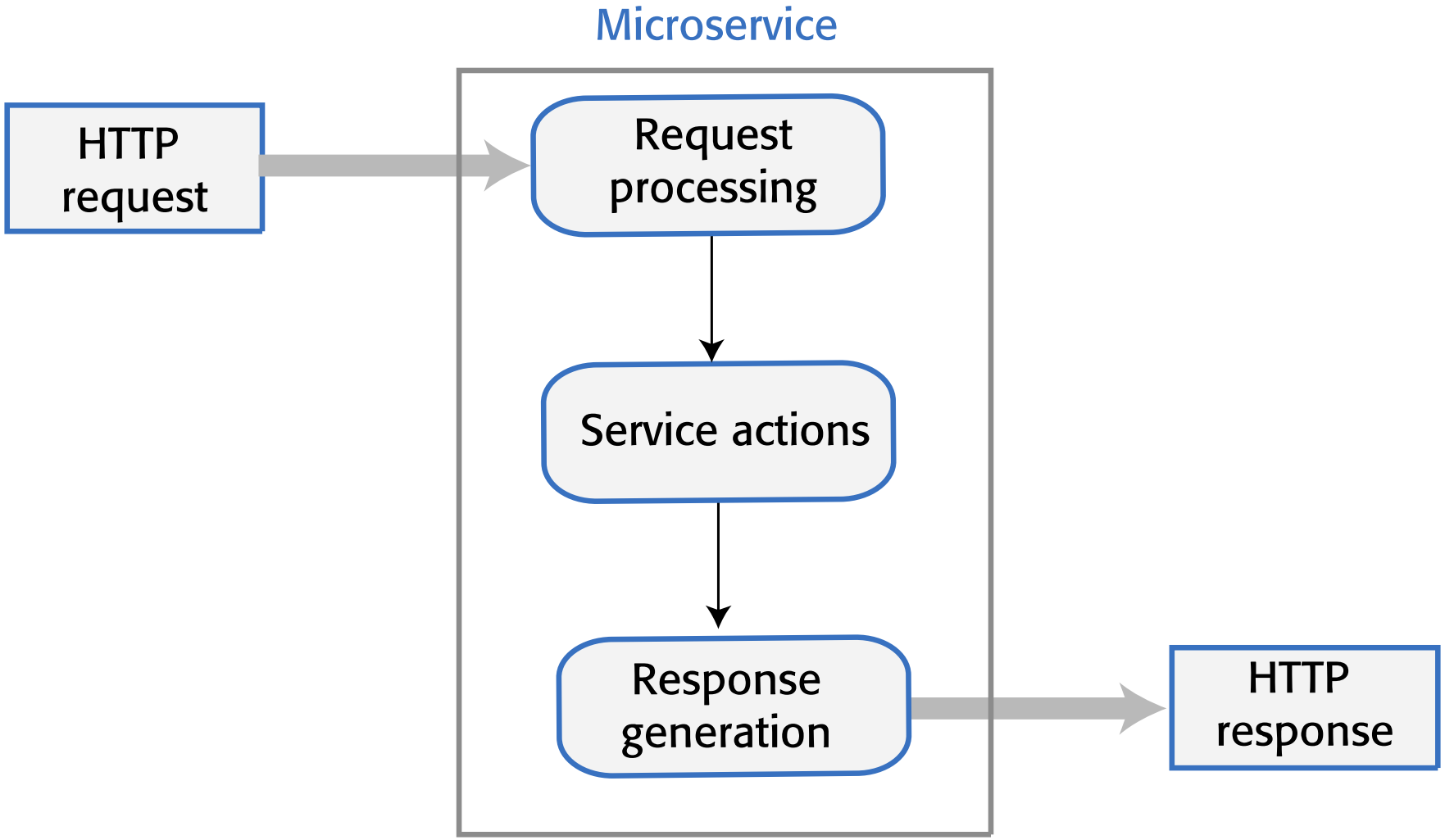
Description des incidents

- ID de l'incident : A90N17061714391
- Date : 17 juin 2017
- Temps rapporté : 1439
- Gravité : Significative
- Description : Bus en panne sur la chaussée nord. Une voie est fermée. Attendez-vous à des retards pouvant aller jusqu'à 30 minutes.

Opérations de service

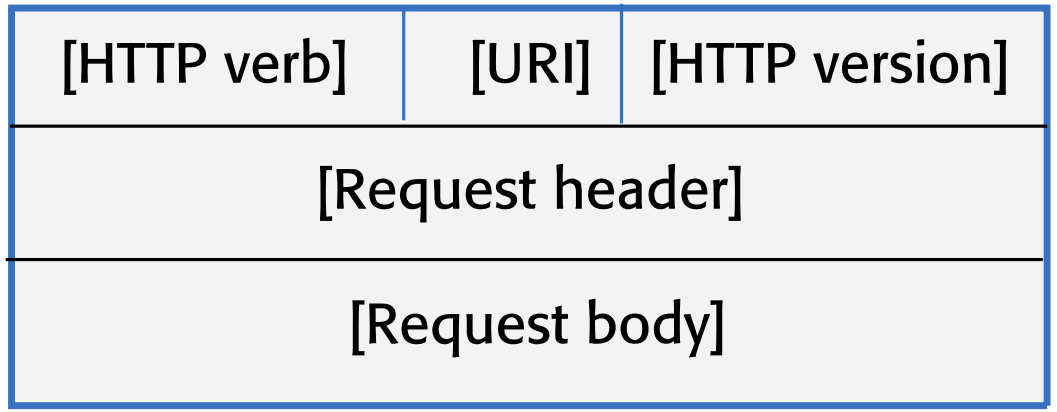
- Récupérer
  - Renvoie des informations sur un ou plusieurs incidents signalés. On y accède en utilisant le verbe GET.
- Ajouter
  - Ajoute des informations sur un nouvel incident. Accès par le verbe POST.
- Mise à jour
  - Met à jour les informations relatives à un incident signalé. Accès par le verbe PUT.
- Supprimer
  - Supprime un incident. Le verbe DELETE est utilisé lorsqu'un incident a été supprimé.

Traitement des requêtes et des réponses HTTP

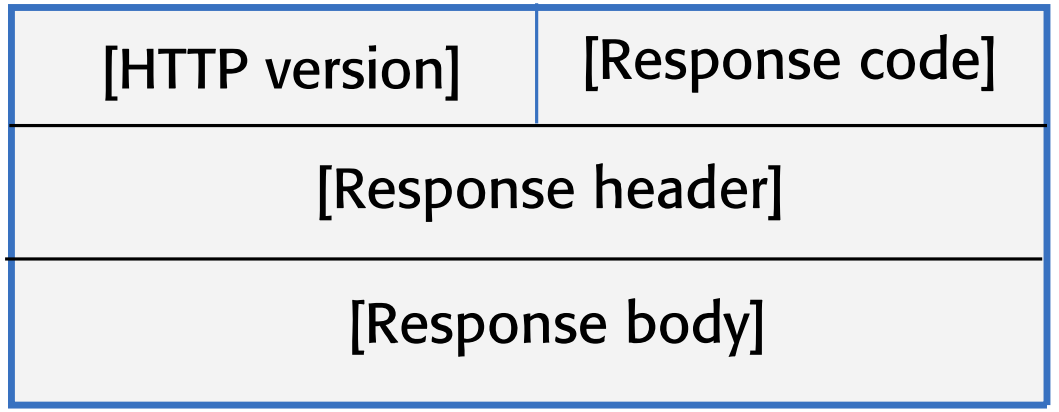


Organisation des messages de requête et de réponse HTTP

REQUEST



RESPONSE



Descriptions XML et JSON

JSON

{ id :
"A90N170617143 91", "date" :
"20170617", "time" :
"1437", "road_id" :
"A90", "place" :
"Stonehaven", "direction" :
"nord", "gravité" :
"significatif", "description" : "Bus en panne sur la chaussée nord. Une voie fermée.
Attendez-vous à des retards pouvant aller jusqu'à 30 minutes" }.

Descriptions XML et JSON

XML

<id&gt ; A90N17061714391 </id&gt ; <date&gt ; 20170617 </date&gt ; <time&gt ; 1437 </time&gt ; ... <description>Bus en panne sur la chaussée nord. Une voie fermée.
Attendez-vous à des retards allant jusqu'à 30 minutes. </description&gt ;

Une requête GET et la réponse associée

REQUEST

GET	incidents/A90/stonehaven/	HTTP/1.1

Host: trafficinfo.net
...
Accept: text/json, text/xml, text/plain
Content-Length: 0

RESPONSE

HTTP/1.1	200

...
Content-Length: 461
Content-Type: text/json

{
"number": "A90N17061714391",
"date": "20170617",
"time": "1437",
"road_id": "A90",
"place": "Stonehaven",
"direction": "north",
"severity": "significant",
"description": "Broken-down bus on north carriageway. One lane closed. Expect delays of up to 30 minutes."
}
{
"number": "A90S17061713001",
"date": "20170617",
"time": "1300",
"road_id": "A90",
"place": "Stonehaven",
"direction": "south",
"severity": "minor",
"description": "Grass cutting on verge. Minor delays"
}

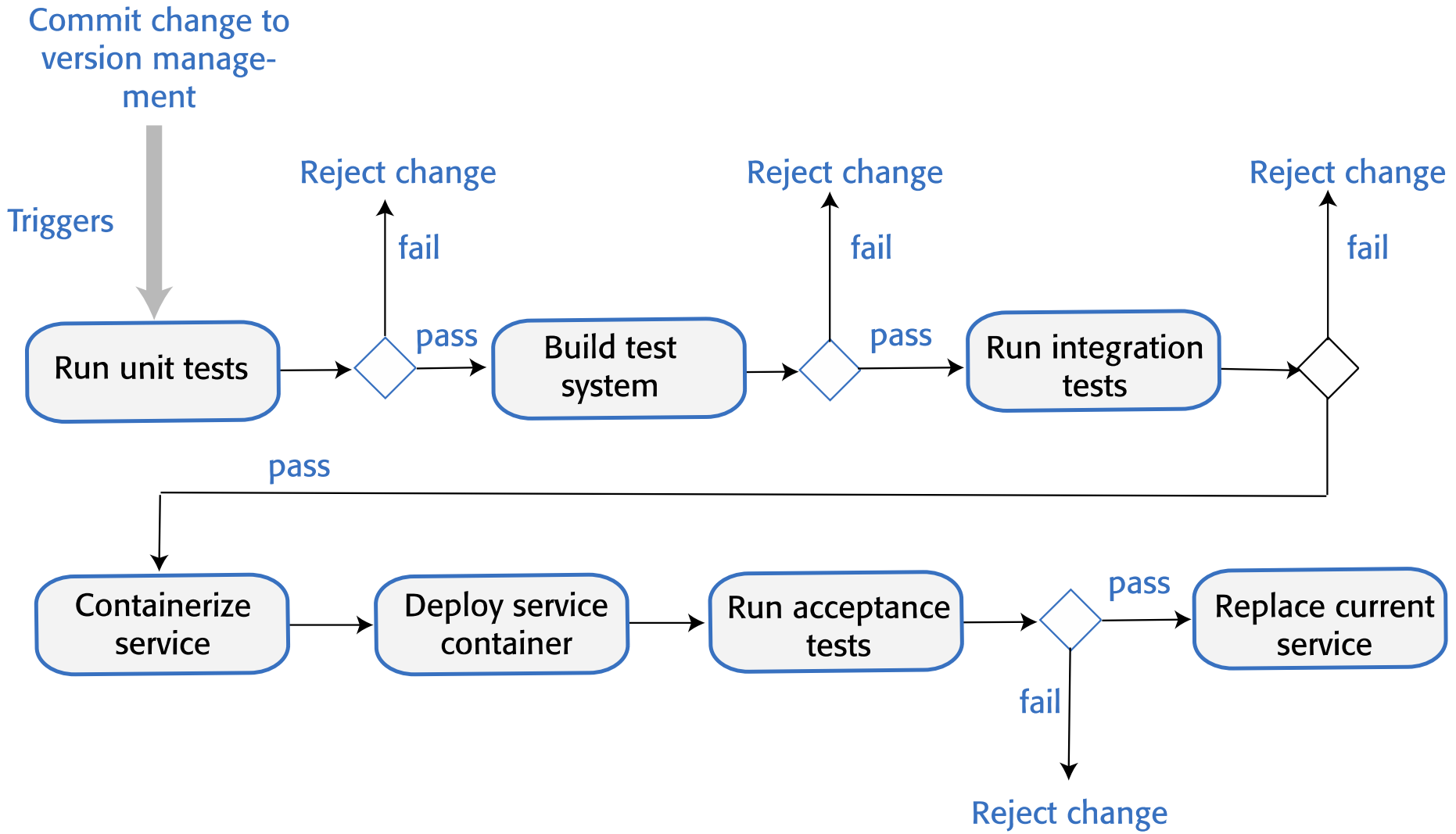
Déploiement des services

- Une fois qu'un système a été développé et livré, il doit être déployé sur des serveurs, surveillé pour détecter les problèmes et mis à jour lorsque de nouvelles versions sont disponibles.
- Lorsqu'un système est composé de dizaines, voire de centaines de microservices, le déploiement du système est plus complexe que pour les systèmes monolithiques.
- Les équipes de développement des services décident du langage de programmation, de la base de données, des bibliothèques et des autres logiciels de soutien à utiliser pour mettre en œuvre leur service. Par conséquent, il n'existe pas de configuration de déploiement "standard" pour tous les services.
- Il est désormais normal que les équipes de développement de microservices soient responsables du déploiement et de la gestion des services, ainsi que du développement des logiciels, et qu'elles utilisent le déploiement continu.
- Le déploiement continu signifie que dès qu'une modification d'un service a été effectuée et validée, le service modifié est redéployé.

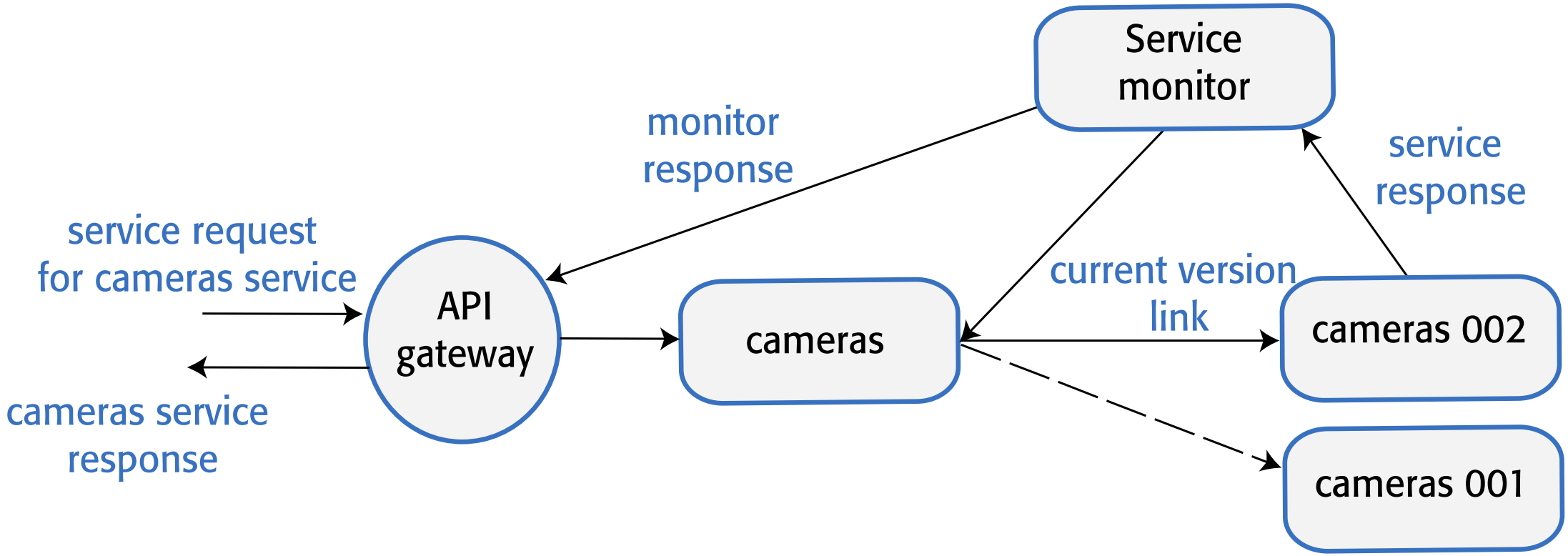
Automatisation du déploiement

- Le déploiement continu dépend de l'automatisation, de sorte que dès qu'une modification est validée, une série d'activités automatisées est déclenchée pour tester le logiciel.
- Si le logiciel "réussit" ces tests, il entre alors dans un autre pipeline d'automatisation qui le conditionne et le déploie.
- Le déploiement d'une nouvelle version d'un service commence par l'enregistrement par le programmeur des modifications du code dans un système de gestion du code tel que Git.
- Cela déclenche un ensemble de tests automatisés qui s'exécutent en utilisant le service modifié. Si tous les tests du service se déroulent avec succès, une nouvelle version du système intégrant le service modifié est créée.
- Une autre série de tests automatisés du système est ensuite exécutée. S'ils se déroulent avec succès, le service est prêt à être déployé.

Un pipeline de déploiement continu



Services versionnés



Points clés 1

- Un microservice est un composant logiciel indépendant et autonome qui s'exécute dans son propre processus et communique avec d'autres microservices à l'aide de protocoles légers.
- Les microservices d'un système peuvent être mis en œuvre à l'aide de différents langages de programmation et technologies de base de données.
- Les microservices ont une responsabilité unique et doivent être conçus de manière à pouvoir être facilement modifiés sans devoir changer d'autres microservices dans le système.
- L'architecture microservices est un style architectural dans lequel le système est construit à partir de microservices communicants. Elle est bien adaptée aux systèmes basés sur l'informatique en nuage, où chaque microservice peut fonctionner dans son propre conteneur.
- Les deux principales responsabilités des architectes d'un système de microservices sont de décider comment structurer le système en microservices et de décider comment les microservices doivent communiquer et être coordonnés.

Points clés 2

- Les décisions relatives à la communication et à la coordination comprennent le choix des protocoles de communication des microservices, le partage des données, la question de savoir si les services doivent être coordonnés de manière centralisée et la gestion des défaillances.
- Le style architectural RESTful est largement utilisé dans les systèmes basés sur les microservices. Les services sont conçus de manière à ce que les verbes HTTP, GET, POST, PUT et DELETE, correspondent aux opérations de service.
- Le style RESTful est basé sur des ressources numériques qui, dans une architecture de microservices, peuvent être représentées à l'aide de XML ou, plus communément, de JSON.
- Le déploiement continu est un processus par lequel de nouvelles versions d'un service sont mises en production dès qu'un changement de service a été effectué. Il s'agit d'un processus entièrement automatisé qui s'appuie sur des tests automatisés pour vérifier que la nouvelle version est de "qualité production".
- Si le déploiement continu est utilisé, il peut être nécessaire de maintenir plusieurs versions des services déployés afin de pouvoir passer à une version plus ancienne si des problèmes sont découverts dans un service nouvellement déployé.