Introduction aux microservices

Plan des cours

Jour 1 : Introduction à l'architecture microservices

Jour 2 : Création d’une mini application CRUD Javascript en microservices

Jour 3 : API Gateway

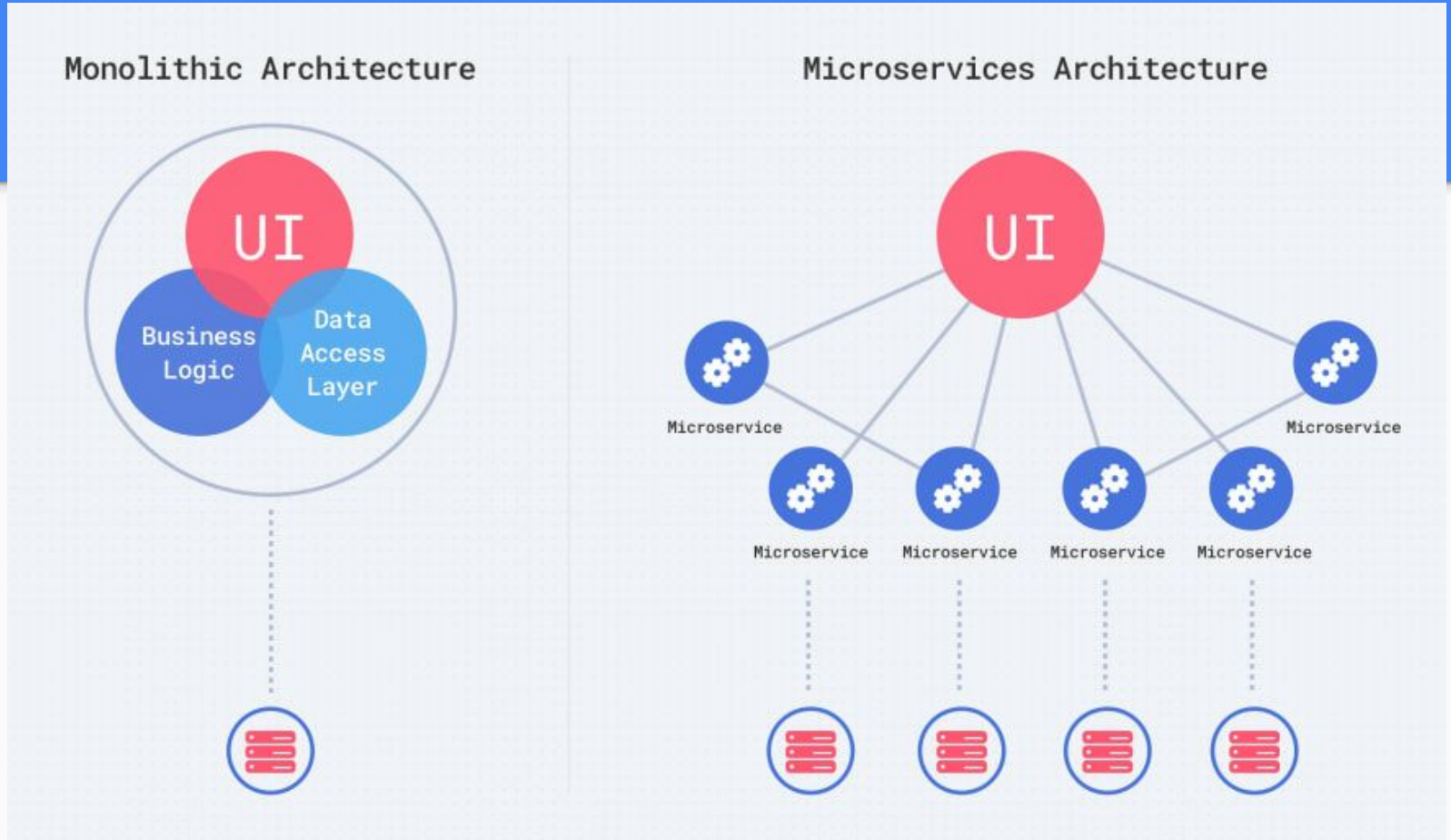
Jour 4 : Event Bus le matin et récap/ressources pour aller plus loin l'après-midi

Qu’est ce que l’architecture microservices ?

L'architecture microservices consiste à décomposer une application en petites briques indépendantes les unes des autres, dans le but d'isoler les fonctions clés. Chacune de ces fonctions est appelée un "service". Ces services sont créés pour répondre à des besoins métier précis et uniques, tels que la gestion des utilisateurs, les paiements, l'envoi d'e-mails ou encore les notifications. De plus, ils sont indépendants et modulables, ce qui signifie que chacun peut être développé et déployé sans affecter les autres.

Ce type d'architecture s'oppose aux architectures monolithiques qui sont construites comme une seule unité autonome. Avec l'architecture microservices, en revanche, nous avons de multiples petites unités indépendantes qui interagissent ensemble pour effectuer les tâches nécessaires.

Schéma Monolithe VS microservices



Différence entre Monolithe/Microservices

Architecture monolithique :

- Structure : Une application monolithique est constituée d'un seul et unique code source qui regroupe toutes les fonctionnalités de l'application.
- Couplage : Les différents composants de l'application sont étroitement couplés et dépendent les uns des autres.
- Déploiement : L'application est déployée en tant qu'entité unique, ce qui rend les déploiements plus complexes et nécessite un redémarrage complet de l'application pour les mises à jour.
- Évolutivité : L'évolutivité est limitée, car l'ensemble de l'application doit être mis à l'échelle, même si seule une partie spécifique nécessite davantage de ressources.
- Maintenance : La maintenance peut être complexe, car chaque modification nécessite la recompilation et le déploiement de l'ensemble de l'application.
- Technologies : Les technologies utilisées dans une architecture monolithique sont généralement homogènes, car tous les composants partagent le même environnement d'exécution.

Architecture basée sur les microservices :

- Structure : Une application basée sur les microservices est constituée de multiples services indépendants qui fonctionnent ensemble pour fournir les fonctionnalités globales de l'application.
- Découplage : Les microservices sont indépendants les uns des autres, avec leurs propres responsabilités et leur propre logique de fonctionnement. Ils peuvent être développés, déployés et mis à l'échelle de manière indépendante.
- Déploiement : Chaque microservice peut être déployé séparément, ce qui facilite les mises à jour continues sans nécessiter le redémarrage de l'ensemble de l'application.
- Évolutivité : L'évolutivité est plus fine-grain, car seuls les microservices spécifiques nécessitant plus de ressources peuvent être mis à l'échelle, ce

Avantages
de l’architecture Monolithe

Simplicité de développement : Les architectures monolithiques sont souvent plus simples à concevoir et à développer, car toutes les fonctionnalités sont regroupées dans une seule application.

Facilité de déploiement : Étant donné qu'une application monolithique est déployée comme une seule unité, le déploiement peut être relativement simple.

Moins de complexité opérationnelle : La gestion et l'exploitation d'une seule application sont souvent moins complexes que la gestion de plusieurs services indépendants.

Communication interne plus rapide : Puisque tous les composants de l'application résident dans la même mémoire, la communication entre les différentes parties peut être plus rapide.

Inconvénients de l’architecture Monolithe

Difficulté d'évolutivité : L'évolutivité d'une application monolithique peut être limitée, car l'ensemble de l'application doit être mis à l'échelle même si seules certaines parties ont besoin de ressources supplémentaires.

Couplage fort : Les composants d'une application monolithique sont souvent étroitement couplés, ce qui peut rendre les modifications et les mises à jour plus complexes.

Difficultés de maintenance : Les modifications apportées à une application monolithique nécessitent généralement la recompilation et le redéploiement de l'ensemble de l'application, ce qui peut augmenter le risque d'erreurs et rendre la maintenance plus difficile.

Technologie homogène : Dans une architecture monolithique, tous les composants partagent le même environnement d'exécution et les mêmes technologies, ce qui limite la flexibilité dans le choix des technologies.

Avantages d’une architecture microservices

Évolutivité fine-grain : Les microservices permettent une mise à l'échelle fine-grain, car chaque service peut être mis à l'échelle indépendamment selon les besoins spécifiques, permettant ainsi une utilisation plus efficace des ressources.

Flexibilité technologique : Chaque microservice peut utiliser la technologie et le langage de programmation les plus appropriés pour répondre à ses besoins spécifiques, ce qui offre une plus grande flexibilité dans le choix des technologies.

Facilité de maintenance et d'évolution : Les microservices sont indépendants les uns des autres, ce qui facilite la maintenance et les mises à jour sans affecter l'ensemble du système. Les déploiements sont plus rapides et moins risqués.

Favorise l'agilité et l'autonomie des équipes : Les équipes de développement peuvent travailler de manière autonome sur des microservices spécifiques, ce qui favorise l'agilité et permet des cycles de développement plus rapides.

Inconvénients d’une architecture microservices

Complexité de développement : Les architectures basées sur les microservices peuvent être plus complexes à concevoir, à développer et à gérer en raison du nombre de services, de la gestion des communications et de la coordination entre les services.

Complexité opérationnelle : La gestion d'un environnement avec de nombreux services indépendants peut être plus complexe, nécessitant des outils et une infrastructure supplémentaires pour la gestion, la surveillance et la résilience.

Communication externe : Les microservices doivent communiquer entre eux via des mécanismes tels que des API ou des messages, ce qui peut introduire une latence supplémentaire et nécessiter une gestion plus attentive des erreurs de communication.

Quand utiliser les microservices ?

- Quand la taille et la complexité de votre application le nécessitent.
- Quand vos équipes sont prêtes à franchir le cap.
- Quand les avantages des microservices mentionnés précédemment surpassent leurs inconvénients.

On ne doit pas construire une architecture microservices sans raison valable. Appliquer des microservices à de petites applications n'a pas beaucoup de sens.

Indépendance et découplage

Les microservices sont conçus pour être indépendants les uns des autres, avec des responsabilités clairement définies. Chaque microservice peut être développé, déployé, mis à l'échelle et mis à jour indépendamment des autres.

Cela permet une meilleure flexibilité et agilité dans le développement et la maintenance des services.

Responsabilité unique

Chaque microservice est responsable d'une fonctionnalité spécifique de l'application. Il est conçu pour être hautement cohésif, en se concentrant sur un domaine métier restreint. Cela permet une meilleure modularité et facilite la compréhension, la maintenance et l'évolution du système.

Communication entre microservices

Les microservices communiquent entre eux de différentes façons, par exemple via des API REST ou des messages asynchrones.

C'est de cette façon que les microservices pourront rester autonomes tout en ayant la possibilité de se partager des informations et d'interagir les uns avec les autres.

Lors de prochains cours nous reviendrons plus longuement en détail sur les différentes méthodes possible pour faire communiqué des microservices (API REST, event bus, server mesh etc)

Isolation des données

En règle général chaque microservice a sa propre base de données ou utilise des mécanismes de communication asynchrone pour partager des données avec d'autres services. Cela garantit l'isolation des données entre les services et permet une meilleure évolutivité et résilience.

Déploiement

Les microservices peuvent être déployés indépendamment les uns des autres, ce qui facilite le déploiement continu. De plus, chaque microservice peut être mis à l'échelle de manière indépendante en fonction des besoins de charge, ce qui permet une utilisation plus efficace des ressources.

Gestion de l’architecture distribuée

Les microservices sont souvent déployés dans un environnement distribué, ce qui nécessite une gestion des problèmes tels que la latence réseau, la tolérance aux pannes, la cohérence des données et la sécurité. Des pratiques telles que la surveillance, la gestion des erreurs, la journalisation et la gestion des transactions distribuées sont essentielles pour gérer ces défis.

Automatisation et infrastructure évolutive

L'automatisation joue un rôle clé dans la gestion des microservices. Des outils tels que les conteneurs (par exemple, Docker) et les outils d'orchestration (par exemple, Kubernetes) facilitent le déploiement, la gestion et l'évolutivité des microservices. L'infrastructure doit être flexible et capable de s'adapter aux besoins changeants des microservices.

Monitoring

Le monitoring et l'observabilité des applications microservices consistent à collecter, stocker, visualiser et analyser les données liées aux performances, aux journaux et aux erreurs des microservices.

Cela peut inclure des journaux (logs), des métriques (comme le taux de requêtes, la latence, etc.) et des traces (tracing) pour suivre le parcours des requêtes à travers les différents services.

Cela permet d'assurer le bon fonctionnement de l'application, d'identifier les problèmes et de prendre des mesures pour les résoudre rapidement.

La conteneuriasation dans les microservices

La conteneurisation dans les microservices permet de mettre en place un environnement d'exécution isolé et portable pour chaque service. Voici une explication simple de l'utilité de la conteneurisation dans les microservices :

Imaginez que chaque service de votre application est une boîte individuelle contenant tout ce dont ce service a besoin pour fonctionner, y compris le code, les bibliothèques et les dépendances. Ces boîtes s'appellent des conteneurs.

Maintenant, avec la conteneurisation, vous pouvez emballer chaque service dans son propre conteneur, de manière à ce qu'il fonctionne de manière indépendante des autres services. Chaque conteneur est isolé et n'interfère pas avec les autres.

Docker

Docker est une plateforme open-source qui permet de créer, déployer et exécuter des applications dans des conteneurs. Les conteneurs sont des environnements légers et isolés qui encapsulent une application et toutes ses dépendances, ce qui permet d'assurer la portabilité et la reproductibilité de l'application sur différentes infrastructures.

Docker offre plusieurs avantages pour la mise en œuvre des microservices :

1. Isolation des services : Chaque service peut être encapsulé dans un conteneur Docker indépendant, ce qui permet une isolation complète des dépendances et de l'environnement d'exécution. Cela évite les conflits entre les services et facilite leur déploiement et leur gestion.
2. Portabilité : Les conteneurs Docker offrent une portabilité élevée. Un conteneur Docker peut être créé sur un environnement local et ensuite être déployé de manière cohérente sur n'importe quel autre environnement compatible avec Docker, qu'il s'agisse d'un serveur local, d'un cloud public ou d'un cluster d'orchestration comme Kubernetes.
3. Gestion des dépendances : Docker facilite la gestion des dépendances des microservices. Chaque service peut inclure ses propres dépendances spécifiques dans son conteneur Docker, garantissant ainsi la compatibilité des versions des bibliothèques et des outils utilisés par le service.
4. Déploiement et mise à l'échelle facilités : Docker simplifie le déploiement des microservices en permettant leur encapsulation dans des conteneurs autonomes. Les conteneurs peuvent être déployés et orchestrés facilement sur différentes machines physiques ou virtuelles, ce qui facilite la mise à l'échelle horizontale des services en fonction des besoins de charge.
5. Gestion des versions et des mises à jour : Avec Docker, il est possible de gérer facilement les différentes versions des microservices. Chaque version peut être encapsulée dans un conteneur Docker distinct, permettant ainsi de déployer et de tester facilement les nouvelles versions tout en conservant les anciennes versions en production.
6. Facilité de collaboration : Docker facilite la collaboration entre les équipes de développement et d'exploitation. Les développeurs peuvent travailler sur des environnements de développement cohérents et partager facilement les images Docker de leurs services. Les opérations peuvent déployer ces images sur différents environnements sans se soucier des dépendances ou des configurations spécifiques.

En résumé, Docker offre une solution pratique pour la gestion des microservices en fournissant une isolation, une portabilité, une gestion des dépendances et une

Kubernetes

Kubernetes est un outil d'orchestration de conteneurs qui fonctionne en tandem avec Docker dans une architecture de microservices. Voici une explication simple de l'utilité de Kubernetes :

Imaginez que Docker est une boîte qui contient votre application et toutes ses dépendances. Docker permet d'emballer votre application dans un conteneur, ce qui le rend portable et facile à déployer sur différentes machines.

Maintenant, imaginez que vous avez plusieurs de ces boîtes Docker contenant différents services de votre application dans un environnement de microservices. Kubernetes agit comme un chef d'orchestre qui gère ces boîtes Docker et assure leur bon fonctionnement.

Atelier pratique en groupe

Faisons de la pédagogie active ! Découvrez les notions importantes des microservices en effectuant vos propres recherches en groupe et en présentant vos travaux sous forme de slides.

Groupe 1 Orchestration et Chorégraphie dans les microservices

Groupe 2 Message Queuing

Groupe 3 Server Mesh

Groupe 4 Event-Bus

Groupe 5 Event-Driven architecture

Groupe 6 Domain Driven Design et microservices

Groupe 7 API Gateway

Fin de la première journée !

À demain pour la mise en pratique avec la création de notre propre mini-application architecturée sous forme de microservices !