Modèles de conception, architectures et implémentations du cloud

AWS Directives prescriptives

AWS Directives prescriptives: Modèles de conception, architectures et implémentations du cloud

Table of Contents

Modèles de conception, architectures et implémentations du cloud

Anitha Deenadayalan, Amazon Web Services (AWS)

Mai 2024 (historique du document)

Ce guide fournit des conseils pour la mise en œuvre de modèles de conception de modernisation couramment utilisés à l'aide de AWS services. De plus en plus d'applications modernes sont conçues à l'aide d'architectures de microservices afin de garantir la capacité de mise à l'échelle, d'améliorer la rapidité de publication, de diminuer l'impact des modifications et de réduire la régression. Cela permet d'améliorer la productivité et l'agilité des développeurs, de renforcer l'innovation et de se concentrer davantage sur les besoins de l'entreprise. Les architectures de microservices prennent également en charge l'utilisation de la meilleure technologie pour le service et la base de données, et favorisent le code polyglotte et la persistance polyglotte.

Traditionnellement, les applications monolithiques s'exécutent selon un processus unique, utilisent un seul magasin de données et s'exécutent sur des serveurs qui se mettent à l'échelle verticalement. En comparaison, les applications de microservices modernes sont précises, possèdent des domaines de défaillance indépendants, s'exécutent en tant que services sur le réseau et peuvent utiliser plusieurs magasins de données en fonction du cas d'utilisation. Les services se mettent à l'échelle horizontalement et une seule transaction peut couvrir plusieurs bases de données. Les équipes de développement doivent se concentrer sur la communication réseau, la persistance polyglotte, la mise à l'échelle horizontale, la cohérence éventuelle et le traitement des transactions dans les magasins de données lorsqu'elles développent des applications à l'aide d'architectures de microservices. Par conséquent, les modèles de modernisation sont essentiels pour résoudre les problèmes courants du développement d'applications modernes, et ils contribuent à accélérer la livraison des logiciels.

Ce guide fournit une référence technique aux architectes cloud, aux responsables techniques, aux propriétaires d'applications et d'entreprises, ainsi qu'aux développeurs qui souhaitent choisir la bonne architecture cloud pour les modèles de conception en fonction des bonnes pratiques bien conçues. Chaque modèle décrit dans ce guide concerne un ou plusieurs scénarios connus dans le cadre des architectures de microservices. Le guide aborde les problèmes et les considérations associés à chaque modèle, fournit une implémentation architecturale de haut niveau et décrit l'implémentation AWS pour le modèle. GitHub Des exemples open source et des liens vers des ateliers sont fournis lorsqu'ils sont disponibles.

Le guide couvre les modèles suivants :

- Couche anticorruption
- Modèles de routage des API :
  - Routage du nom d'hôte
  - Routage des chemins
  - Routage des en-têtes HTTP
- Disjoncteur de circuit
- Approvisionnement d'événement
- Architecture hexagonale
- Publier/s'abonner
- Réessayer avec rétrogradation
- Modèles de saga :
  - Chorégraphie de saga
  - Orchestration de saga
- Dispersez et collectez
- Figuier étrangleur
- Boîte d'envoi transactionnelle

Résultats commerciaux ciblés

En utilisant les modèles décrits dans ce guide pour moderniser vos applications, vous pouvez :

- Concevoir et implémenter des architectures fiables, sécurisées et efficaces sur le plan opérationnel, optimisées en termes de coûts et de performances.
- Réduire le temps de cycle pour les cas d'utilisation qui nécessitent ces modèles, afin de pouvoir vous concentrer sur les défis propres à l'organisation.
- Accélérer le développement en normalisant les implémentations de modèles à l'aide des services AWS.
- Aider vos développeurs à créer des applications modernes sans avoir à hériter de dettes techniques.

Modèle de couche anticorruption

Intention

Le modèle de couche anticorruption (ACL) agit comme une couche de médiation qui traduit la sémantique du modèle de domaine d'un système à un autre. Il traduit le modèle du contexte délimité en amont (monolithe) en un modèle adapté au contexte délimité en aval (microservice) avant d'utiliser le contrat de communication établi par l'équipe en amont. Ce modèle peut s'appliquer lorsque le contexte délimité en aval contient un sous-domaine principal ou lorsque le modèle en amont est un système existant non modifiable. Elle réduit également les risques liés à la transformation et à l'interruption des activités en empêchant toute modification des appelants lorsque leurs appels doivent être redirigés de manière transparente vers le système cible.

Motivation

Au cours du processus de migration, lorsqu'une application monolithique est migrée vers des microservices, des modifications peuvent survenir dans la sémantique du modèle de domaine du service nouvellement migré. Lorsque les fonctionnalités du monolithe sont requises pour appeler ces microservices, les appels doivent être routés vers le service migré sans nécessiter de modification des services appelants. Le modèle ACL permet au monolithe d'appeler les microservices de manière transparente en agissant comme un adaptateur ou une couche de façade qui traduit les appels selon la nouvelle sémantique.

Applicabilité

Envisagez d'utiliser ce modèle lorsque :

- Votre application monolithique existante doit communiquer avec une fonction qui a été migrée vers un microservice, et le modèle et la sémantique du domaine de service migré diffèrent de la fonctionnalité d'origine.
- Deux systèmes ont une sémantique différente et doivent échanger des données, mais il n'est pas pratique de modifier un système pour qu'il soit compatible avec l'autre système.
- Vous souhaitez utiliser une approche rapide et simplifiée pour adapter un système à un autre avec un impact minimal.

- Votre application communique avec un système externe.

Enjeux et considérations

- Dépendances entre les équipes :Lorsque différents services d'un système appartiennent à différentes équipes, la nouvelle sémantique du modèle de domaine dans les services migrés peut entraîner des modifications dans les systèmes appelants. Toutefois, les équipes peuvent ne pas être en mesure d'apporter ces modifications de manière coordonnée, car elles peuvent avoir d'autres priorités. L'ACL découple les appelés et traduit les appels pour qu'ils correspondent à la sémantique des nouveaux services, évitant ainsi aux appelants d'apporter des modifications au système actuel.
- Frais généraux opérationnels :Le modèle ACL nécessite des efforts supplémentaires pour fonctionner et maintenir. Ce travail inclut l'intégration de l'ACL aux outils de surveillance et d'alerte, au processus de publication et aux processus d'intégration continue et de livraison continue (CI/CD).
- Point de défaillance unique :Toute défaillance de l'ACL peut rendre le service cible inaccessible et provoquer des problèmes d'application. Pour atténuer ce problème, vous devez intégrer des fonctionnalités de nouvelle tentative et des disjoncteurs. Voir leréessayer avec backoffetdisjoncteurmodèles pour mieux comprendre ces options. La configuration d'alertes et de journalisation appropriées améliorera le délai moyen de résolution (MTTR).
- Dette technique :Dans le cadre de votre stratégie de migration ou de modernisation, déterminez si l'ACL sera une solution transitoire ou intérimaire, ou une solution à long terme. S'il s'agit d'une solution provisoire, vous devez enregistrer l'ACL en tant que dette technique et la désactiver une fois que tous les appelants dépendants ont été migrés.
- Latence :La couche supplémentaire peut introduire une latence en raison de la conversion des demandes d'une interface à une autre. Nous vous recommandons de définir et de tester la tolérance de performance dans les applications sensibles au temps de réponse avant de déployer l'ACL dans des environnements de production.
- Obstacle de mise à l'échelle :Dans les applications à forte charge où les services peuvent évoluer jusqu'à atteindre des pics de charge, l'ACL peut devenir un goulot d'étranglement et entraîner des problèmes de mise à l'échelle. Si le service cible évolue à la demande, vous devez concevoir l'ACL pour qu'il évolue en conséquence.
- Implémentation partagée ou spécifique à un service :Vous pouvez concevoir l'ACL comme un objet partagé pour convertir et rediriger les appels vers plusieurs services ou des classes spécifiques

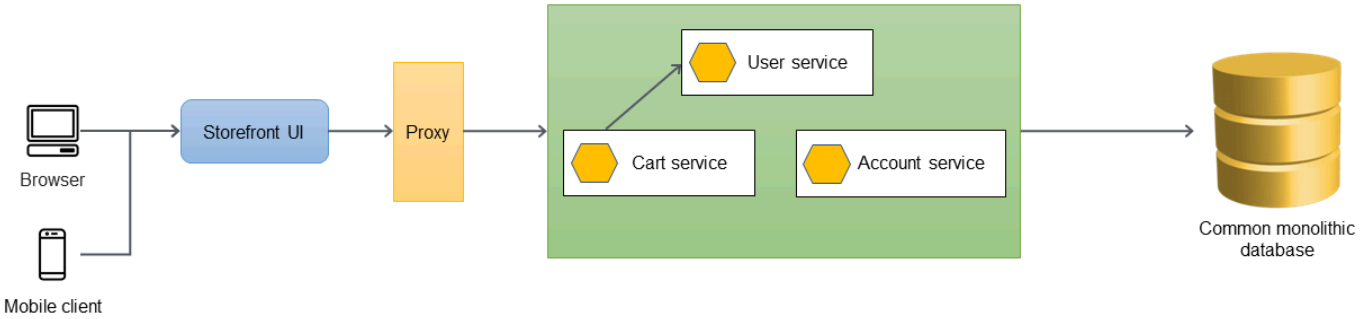
à un service. Tenez compte de la latence, de la mise à l'échelle et de la tolérance aux pannes lorsque vous déterminez le type d'implémentation pour l'ACL.

Mise en œuvre

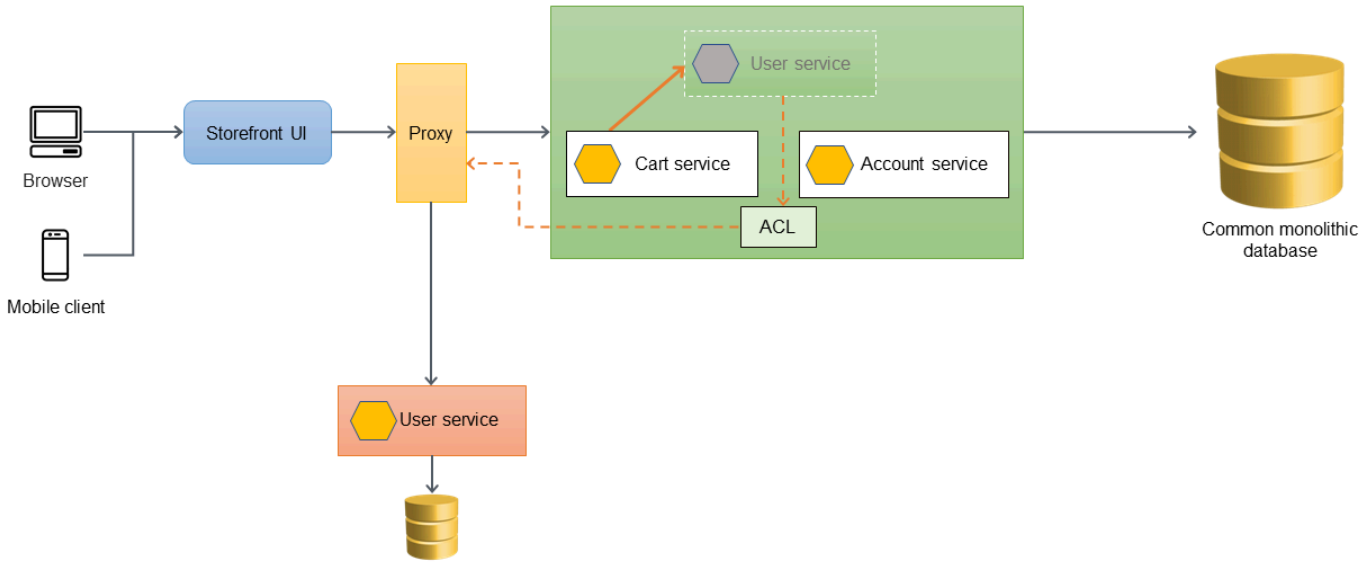
Vous pouvez implémenter l'ACL dans votre application monolithique en tant que classe spécifique au service en cours de migration ou en tant que service indépendant. L'ACL doit être mis hors service une fois que tous les services dépendants ont été migrés vers l'architecture de microservices.

Architecture de haut niveau

Dans l'exemple d'architecture suivant, une application monolithique possède trois services : le service utilisateur, le service du panier et le service des comptes. Le service de panier dépend du service utilisateur et l'application utilise une base de données relationnelle monolithique.

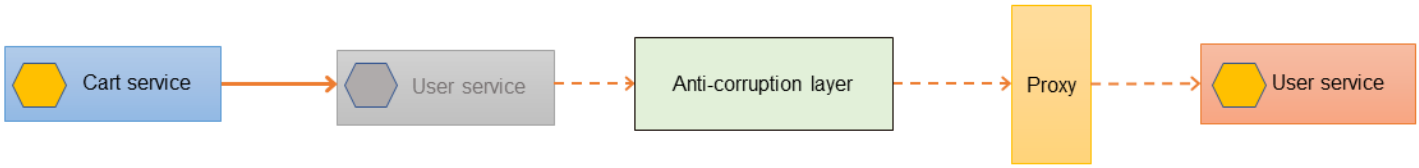


Dans l'architecture suivante, le service utilisateur a été migré vers un nouveau microservice. Le service de panier appelle le service utilisateur, mais l'implémentation n'est plus disponible dans le monolithe. Il est également probable que l'interface du service nouvellement migré ne corresponde pas à son interface précédente, lorsqu'il se trouvait dans l'application monolithique.



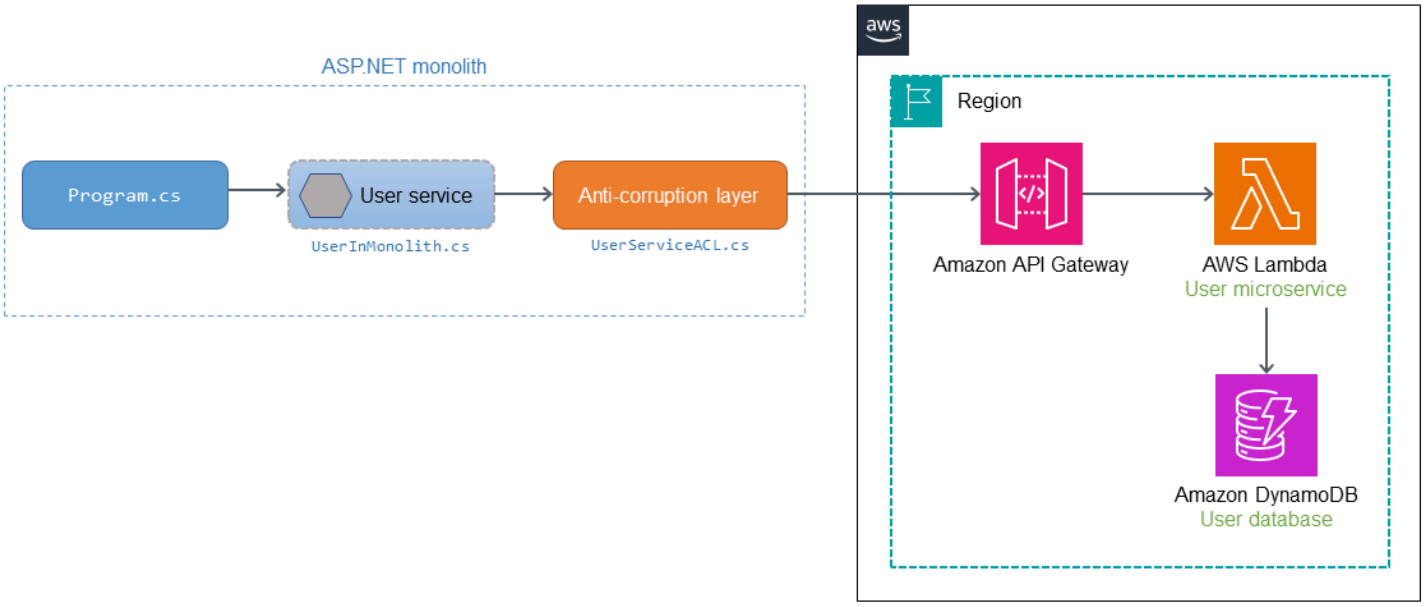
Si le service de panier doit appeler directement le service utilisateur récemment migré, cela nécessitera des modifications du service de panier et des tests approfondis de l'application monolithique. Cela peut accroître le risque de transformation et d'interruption des activités. L'objectif doit être de minimiser les modifications apportées aux fonctionnalités existantes de l'application monolithique.

Dans ce cas, nous vous recommandons d'introduire une ACL entre l'ancien service utilisateur et le service utilisateur récemment migré. L'ACL fonctionne comme un adaptateur ou une façade qui convertit les appels vers la nouvelle interface. L'ACL peut être implémentée dans l'application monolithique en tant que classe (par exemple,`UserServiceFacade`ou`UserServiceAdapter`) qui est spécifique au service qui a été migré. La couche anticorruption doit être mise hors service une fois que tous les services dépendants ont été migrés vers l'architecture de microservices.



Mise en œuvre en utilisantAWSservices

Le schéma suivant montre comment vous pouvez implémenter cet exemple d'ACL en utilisantAWSservices.



Le microservice utilisateur est migré hors de l'application monolithique ASP.NET et déployé en tant queAWS Lambdafonctionne sur AWS. Les appels à la fonction Lambda sont acheminés viaPasserelle d'API Amazon. L'ACL est déployée dans le monolithe pour traduire l'appel afin de l'adapter à la sémantique du microservice utilisateur.

Quand`Program.cs`appelle le service utilisateur (`UserInMonolith.cs`) à l'intérieur du monolithe, l'appel est acheminé vers l'ACL (`UserServiceACL.cs`). L'ACL traduit l'appel vers la nouvelle sémantique et la nouvelle interface, et appelle le microservice via le point de terminaison API Gateway. L'appelant (`Program.cs`) n'est pas au courant de la traduction et du routage qui ont lieu dans le service utilisateur et l'ACL. Comme l'appelant n'est pas au courant des modifications apportées au code, il y a moins de perturbations opérationnelles et moins de risques de transformation.

Exemple de code

L'extrait de code suivant fournit les modifications apportées au service d'origine et l'implémentation de`UserServiceACL.cs`. Lorsqu'une demande est reçue, le service utilisateur d'origine appelle l'ACL. L'ACL convertit l'objet source pour qu'il corresponde à l'interface du service récemment migré, appelle le service et renvoie la réponse à l'appelant.

```
public class UserInMonolith: IUserInMonolith
{
    private readonly IACL _userServiceACL;
    public UserInMonolith(IACL userServiceACL) => (_userServiceACL) = (userServiceACL);
    public async Task<HttpStatusCode> UpdateAddress(UserDetails userDetails)
```

```
    {
        //Wrap the original object in the derived class
        var destUserDetails = new UserDetailsWrapped("user", userDetails);
        //Logic for updating address has been moved to a microservice
        return await _userServiceACL.CallMicroservice(destUserDetails);
    }
}

public class UserServiceACL: IACL
{
    static HttpClient _client = new HttpClient();
    private static string _apiGatewayDev = string.Empty;

    public UserServiceACL()
    {
        IConfiguration config = new
 ConfigurationBuilder().AddJsonFile(AppContext.BaseDirectory + "../../../
config.json").Build();
        _apiGatewayDev = config["APIGatewayURL:Dev"];
        _client.DefaultRequestHeaders.Accept.Add(new
 MediaTypeWithQualityHeaderValue("application/json"));
    }
    public async Task<HttpStatusCode> CallMicroservice(ISourceObject details)
    {
        _apiGatewayDev +=  "/" + details.ServiceName;
        Console.WriteLine(_apiGatewayDev);

        var userDetails = details as UserDetails;
        var userMicroserviceModel = new UserMicroserviceModel();
        userMicroserviceModel.UserId = userDetails.UserId;
        userMicroserviceModel.Address = userDetails.AddressLine1 + ", " +
 userDetails.AddressLine2;
        userMicroserviceModel.City = userDetails.City;
        userMicroserviceModel.State = userDetails.State;
        userMicroserviceModel.Country = userDetails.Country;

        if (Int32.TryParse(userDetails.ZipCode, out int zipCode))
        {
            userMicroserviceModel.ZipCode = zipCode;
            Console.WriteLine("Updated zip code");
        }
        else
        {
            Console.WriteLine("String could not be parsed.");
```

```
            return HttpStatusCode.BadRequest;
        }

        var jsonString =
 JsonSerializer.Serialize<UserMicroserviceModel>(userMicroserviceModel);
        var payload = JsonSerializer.Serialize(userMicroserviceModel);
        var content = new StringContent(payload, Encoding.UTF8, "application/json");

        var response = await _client.PostAsync(_apiGatewayDev, content);
        return response.StatusCode;
    }
}
```

GitHubréférentiel

Pour une implémentation complète de l'exemple d'architecture pour ce modèle, consultez leGitHubdépôt surhttps://github.com/aws-samples/anti-corruption-layer-pattern.

Contenu connexe

- Motif Strangler Fig
- Schéma de disjoncteur
- Réessayez avec le schéma de désactivation

Modèles de routage des API

Dans les environnements de développement agiles, les équipes autonomes (par exemple les escadrons et les tribus) possèdent un ou plusieurs services qui incluent de nombreux microservices. Les équipes présentent ces services sous forme d'API pour permettre à leurs consommateurs d'interagir avec leur groupe de services et d'actions.

Il existe trois méthodes principales pour exposer les API HTTP aux utilisateurs en amont en utilisant des noms d'hôtes et des chemins :

Method	Description	Exemple
Routage du nom d'hôte	Exposez chaque service sous forme de nom d'hôte.	`billing.api.example.com`
Routage des chemins	Exposez chaque service sous forme de chemin.	`api.example.com/billing`
Routage basé sur les en-têtes	Exposez chaque service sous forme d'en-tête HTTP.	`x-example-action: something`

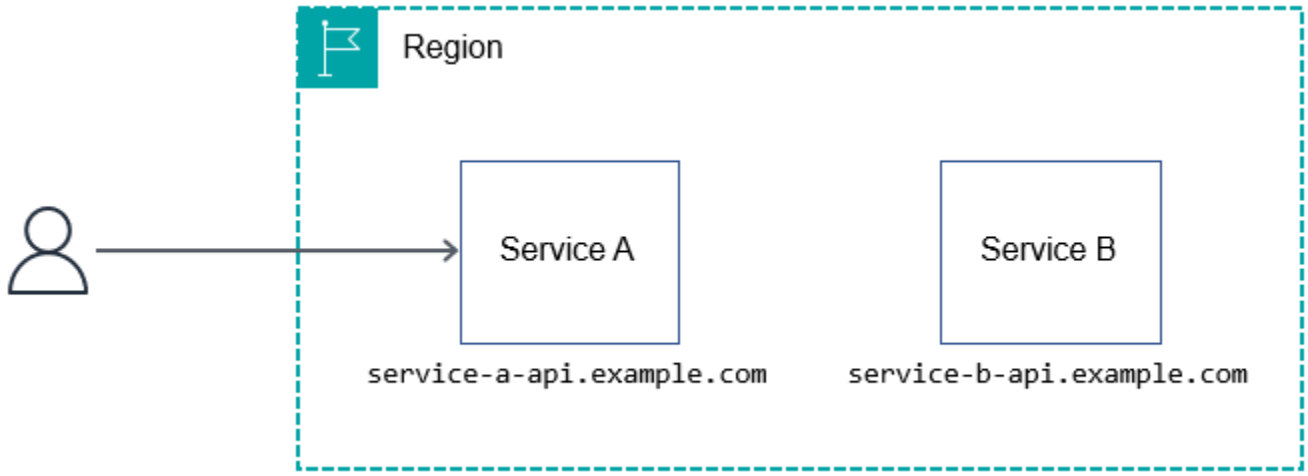
Cette section décrit les cas d'utilisation typiques de ces trois méthodes de routage et leurs compromis pour vous aider à choisir la méthode la mieux adaptée à vos besoins et à votre structure organisationnelle.

Schéma de routage du nom d'hôte

Le routage par nom d'hôte est un mécanisme permettant d'isoler les services d'API en attribuant à chaque API son propre nom d'hôte ; par exemple, `service-a.api.example.com` ou `service-a.example.com`.

Cas d'utilisation types

Le routage à l'aide de noms d'hôtes réduit les frictions lors des lancements, car rien n'est partagé entre les équipes de service. Les équipes sont chargées de tout gérer, des entrées DNS aux opérations de service en production.



Avantages

Le routage par nom d'hôte est de loin la méthode la plus simple et la plus évolutive pour le routage d'API HTTP. Vous pouvez utiliser n'importe quel AWS service approprié pour créer une architecture qui suit cette méthode. Vous pouvez créer une architecture avec Amazon API Gateway AWS AppSync, les Application Load Balancers et Amazon Elastic Compute Cloud (Amazon EC2), ou tout autre service compatible HTTP.

Les équipes peuvent utiliser le routage par nom d'hôte pour être entièrement propriétaires de leur sous-domaine. Cela facilite également l'isolation, le test et l'orchestration de déploiements pour des versions Régions AWS ou des versions spécifiques, `region.service-a.api.example.com` par exemple ou. `dev.region.service-a.api.example.com`

Inconvénients

Lorsque vous utilisez le routage par nom d'hôte, vos clients doivent mémoriser différents noms d'hôte pour interagir avec chaque API que vous exposez. Vous pouvez atténuer ce problème en fournissant un kit SDK client. Cependant, les kits SDK clients présentent leurs propres défis. Par exemple, ils doivent prendre en charge les mises à jour propagées, le multilinguisme, la gestion des versions, la communication des modifications majeures causées par des problèmes de sécurité ou des corrections de bogues, la documentation, etc.

Lorsque vous utilisez le routage par nom d'hôte, vous devez également enregistrer le sous-domaine ou le domaine chaque fois que vous créez un nouveau service.

Schéma de routage des chemins

Le routage par chemins est le mécanisme qui consiste à regrouper plusieurs ou toutes les API sous le même nom d'hôte et à utiliser un URI de demande pour isoler les services ; par exemple `api.example.com/service-a` ou `api.example.com/service-b`.

Cas d'utilisation types

La plupart des équipes optent pour cette méthode, car elles souhaitent une architecture simple : un développeur ne doit mémoriser qu'une seule URL, par exemple `api.example.com`, pour interagir avec l'API HTTP. La documentation des API est souvent plus facile à intégrer, car elle est souvent conservée en un même endroit au lieu d'être répartie sur différents portails ou fichiers PDF.

Le routage basé sur le chemin est considéré comme un mécanisme simple de partage d'une API HTTP. Cependant, il implique des frais opérationnels tels que la configuration, les autorisations, les intégrations et une latence supplémentaire due aux sauts multiples. Cela nécessite également des processus de gestion des modifications matures pour garantir qu'une mauvaise configuration ne perturbe pas tous les services.

Oui AWS, il existe plusieurs manières de partager une API et de l'acheminer efficacement vers le bon service. Les sections suivantes présentent trois approches : le service HTTP, le proxy inverse, API Gateway et Amazon CloudFront. Aucune des approches proposées pour unifier les services d'API ne repose sur l'exécution des services en aval sur AWS. Les services peuvent fonctionner n'importe où sans problème ou sur n'importe quelle technologie, à condition qu'ils soient compatibles avec le protocole HTTP.

Proxy inverse de service HTTP

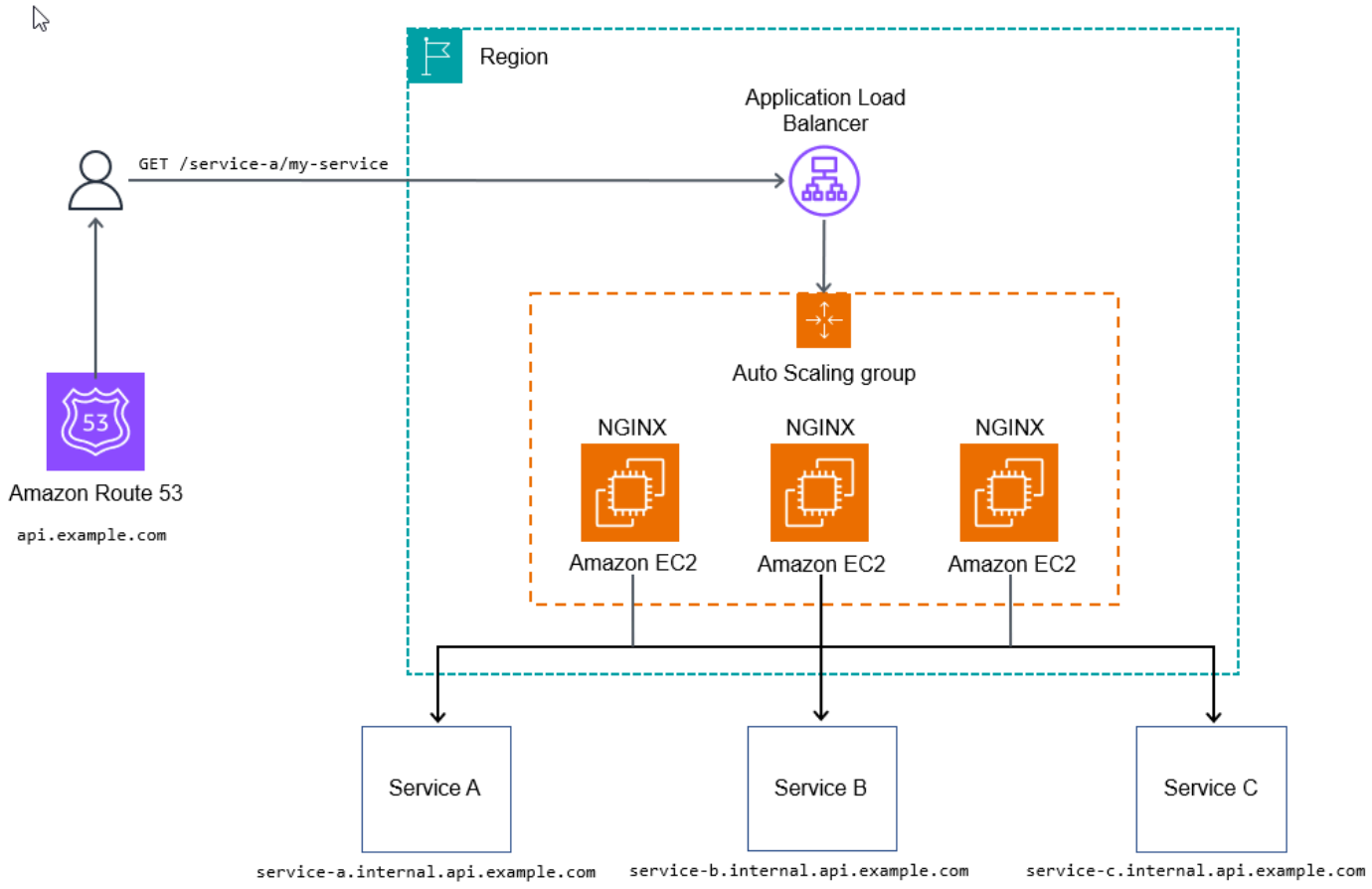
Vous pouvez utiliser un serveur HTTP tel que NGINX pour créer des configurations de routage dynamiques. Dans une architecture Kubernetes, vous pouvez également créer une règle d'entrée correspondant à un chemin d'accès à un service. (Ce guide ne couvre pas l'entrée dans Kubernetes ; veuillez consulter la documentation de Kubernetes pour plus d'informations.)

La configuration suivante de NGINX mappe dynamiquement une demande HTTP de `api.example.com/my-service/` vers `my-service.internal.api.example.com`.

```
server {
    listen  80;
```

```
    location (^/[\w-]+)/(.*) {
        proxy_pass $scheme://$1.internal.api.example.com/$2;
    }
}
```

Le schéma suivant illustre la méthode du proxy inverse de service HTTP.



Cette approche peut être suffisante pour certains cas d'utilisation qui n'utilisent pas de configurations supplémentaires pour commencer à traiter les demandes, ce qui permet à l'API en aval de collecter des métriques et des journaux.

Pour être prêt à être opérationnel en production, vous devez être en mesure d'ajouter de l'observabilité à chaque niveau de votre pile, d'ajouter une configuration supplémentaire ou d'ajouter des scripts pour personnaliser votre point d'entrée d'API afin de permettre des fonctionnalités plus avancées telles que la limitation du débit ou les jetons d'utilisation.

Avantages

L'objectif ultime de la méthode de proxy inverse de service HTTP est de créer une approche évolutive et gérable pour unifier les API dans un domaine unique afin qu'elles apparaissent cohérentes pour tout utilisateur d'API. Cette approche permet également à vos équipes de service de déployer et de gérer leurs propres API, avec un minimum de frais après le déploiement. AWS les services gérés pour le traçage, tels que AWS X-Rayou AWS WAF, sont toujours applicables ici.

Inconvénients

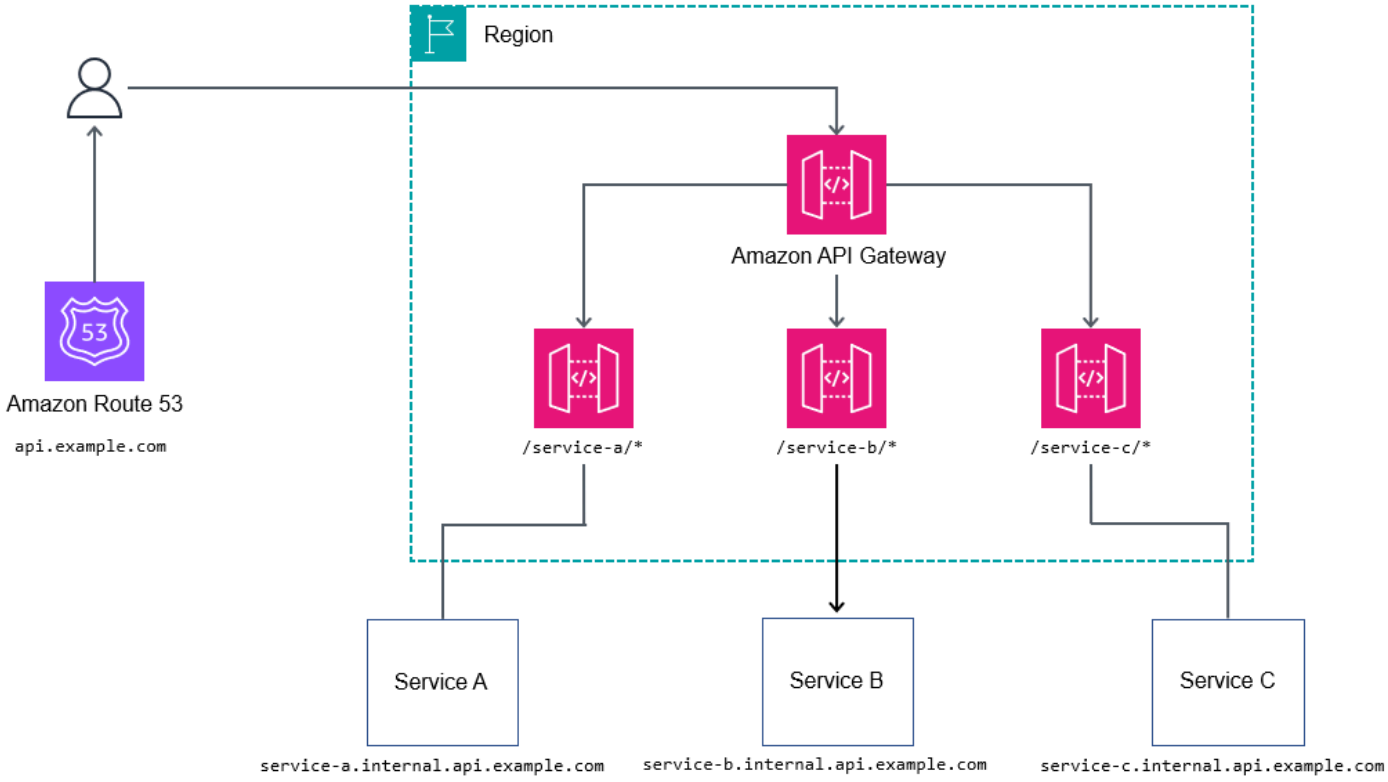
Le principal inconvénient de cette approche réside dans les tests et la gestion approfondis des composants d'infrastructure nécessaires, bien que cela ne pose peut-être aucun problème si vous disposez d'équipes d'ingénierie de fiabilité des sites (SRE) en place.

Cette méthode comporte un point de bascule en matière de coûts. Pour des volumes faibles à moyens, elle est plus coûteuse que certaines des autres méthodes décrites dans ce guide. Pour des volumes élevés, elle est très rentable (environ 100 000 transactions par seconde ou mieux).

API Gateway

Le service Amazon API Gateway (API REST et API HTTP) peut acheminer le trafic d'une manière similaire à la méthode de proxy inverse de service HTTP. L'utilisation d'une passerelle API en mode proxy HTTP fournit un moyen simple de regrouper de nombreux services dans un point d'entrée vers le sous-domaine de premier niveau `api.example.com`, puis de transmettre des demandes proxy au service imbriqué, par exemple `billing.internal.api.example.com`.

Vous ne voulez probablement pas être trop précis en mappant chaque chemin de chaque service dans la passerelle d'API racine ou principale. Optez plutôt pour des chemins génériques, par exemple `/billing/*`, pour transférer les demandes au service de facturation. En ne mappant pas tous les chemins de la passerelle d'API racine ou principale, vous gagnez en flexibilité sur vos API, car vous n'avez pas à mettre à jour la passerelle d'API racine à chaque modification d'API.



Avantages

Pour contrôler des flux de travail plus complexes, tels que la modification des attributs des demandes, les API REST utilisent le langage Apache Velocity Template (VTL) pour vous permettre de modifier la demande et la réponse. Les API REST peuvent apporter des avantages supplémentaires tels que les suivants :

- Authentification N/Z avec AWS Identity and Access Management (IAM), Amazon Cognito ou des autorisateurs AWS Lambda
- AWS X-Ray pour le traçage
- Intégration avec AWS WAF
- Limitation du taux de base
- Utilisation de jetons pour compartimenter les consommateurs en différents niveaux (veuillez consulter Limiter les demandes d'API pour améliorer le débit dans la documentation d'API Gateway)

Inconvénients

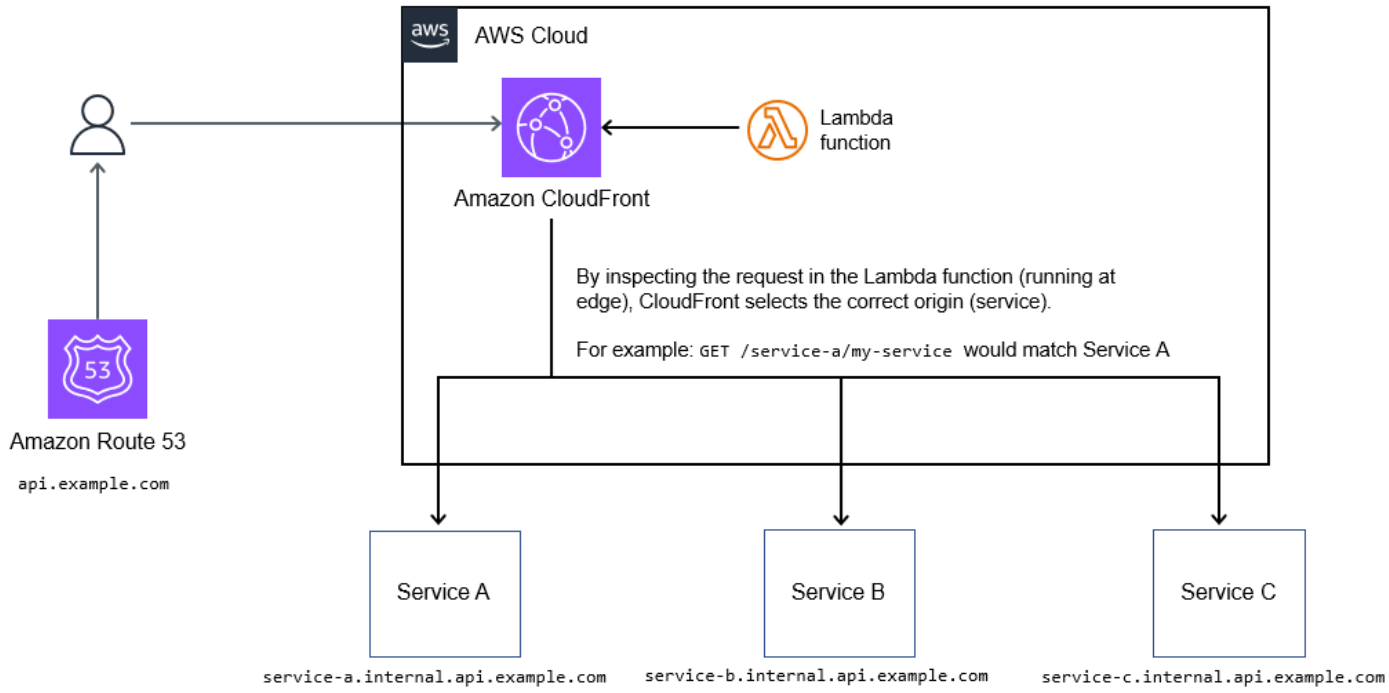
Pour des volumes élevés, le coût peut être un problème pour certains utilisateurs.

CloudFront

Vous pouvez utiliser la fonction de sélection dynamique de l'origine d'Amazon CloudFront pour sélectionner de manière conditionnelle une origine (un service) afin de transférer la demande. Vous pouvez utiliser cette fonctionnalité pour acheminer un certain nombre de services via un seul nom d'hôte, tel que `api.example.com`.

Cas d'utilisation types

La logique de routage fonctionne sous forme de code au sein de la fonction Lambda@Edge. Elle prend donc en charge des mécanismes de routage hautement personnalisables tels que les tests A/B, les versions canary, le signalement des fonctionnalités et la réécriture de chemins. Le diagramme suivant en est l'illustration.



Avantages

Si vous avez besoin de mettre en cache les réponses des API, cette méthode est un bon moyen d'unifier un ensemble de services derrière un seul point de terminaison. Il s'agit d'une méthode rentable pour unifier les collections d'API.

Il prend également CloudFront en charge le chiffrement au niveau du champ ainsi que l'intégration avec les listes de contrôle d'accès de base et AWS WAF les listes de contrôle d'accès de base.

Inconvénients

Cette méthode prend en charge un maximum de 250 origines (services) qui peuvent être unifiées. Cette limite est suffisante pour la plupart des déploiements, mais elle peut entraîner des problèmes avec un grand nombre d'API à mesure que vous développez votre portefeuille de services.

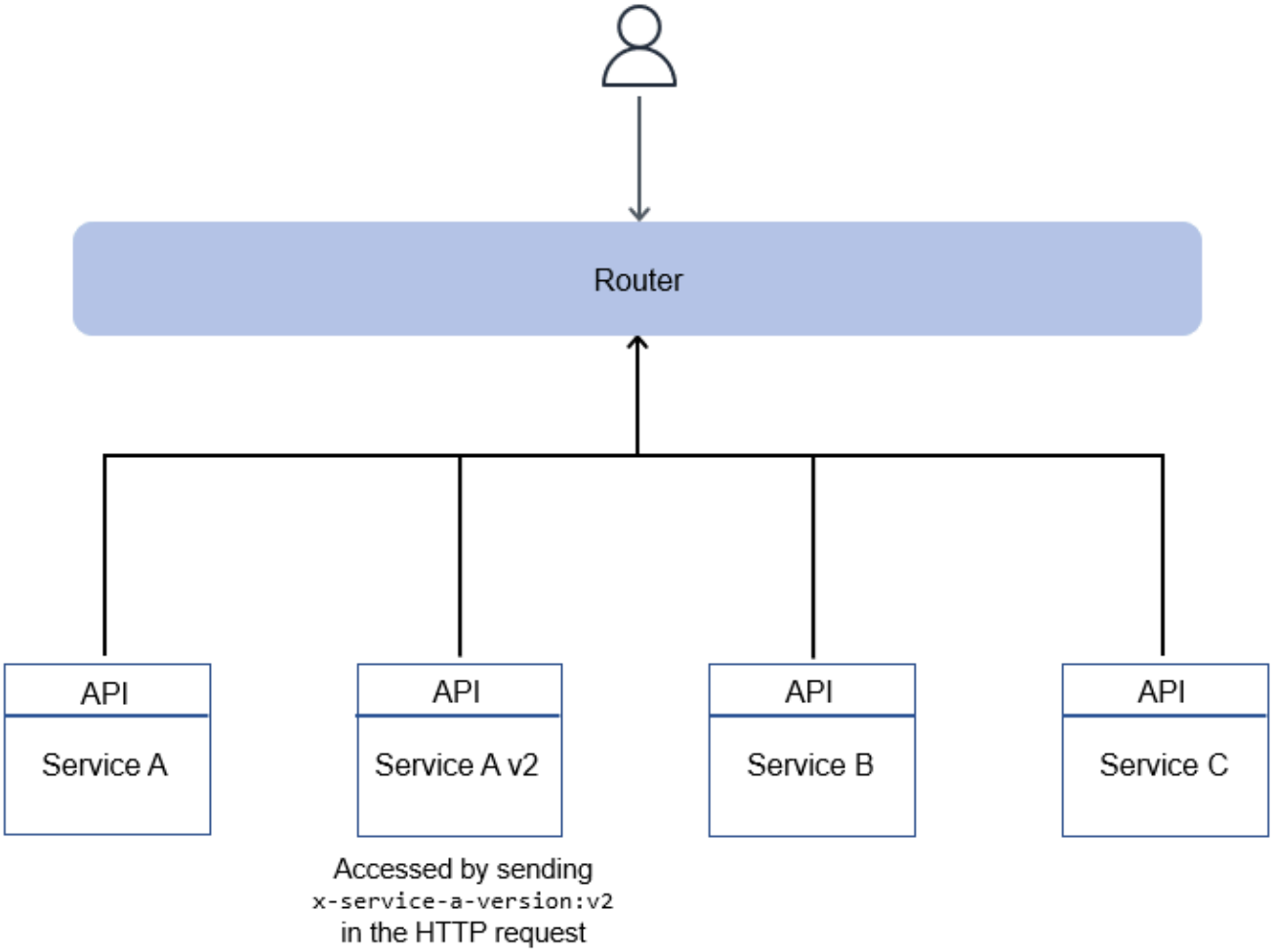
La mise à jour des fonctions Lambda @Edge prend actuellement quelques minutes. CloudFront il faut également jusqu'à 30 minutes pour terminer la propagation des modifications à tous les points de présence. Cela bloque finalement les mises à jour ultérieures jusqu'à ce qu'elles soient terminées.

Modèle de routage des en-têtes HTTP

Le routage basé sur les en-têtes vous permet de cibler le service approprié pour chaque demande en spécifiant un en-tête HTTP dans la requête HTTP. Par exemple, l'envoi de l'en-tête `x-service-a-action: get-thing` vous permettra de `get thing` à partir de `Service A`. Le chemin de la demande est toujours important, car il fournit des indications sur la ressource sur laquelle vous essayez de travailler.

En plus d'utiliser le routage des en-têtes HTTP pour les actions, vous pouvez l'utiliser comme mécanisme pour le routage des versions, pour activer les indicateurs de fonctionnalité, les tests A/B ou pour des besoins similaires. En réalité, vous utiliserez probablement le routage des en-têtes avec l'une des autres méthodes de routage pour créer des API robustes.

L'architecture du routage des en-têtes HTTP comporte généralement une fine couche de routage devant les microservices qui achemine vers le service approprié et renvoie une réponse, comme illustré dans le schéma suivant. Cette couche de routage peut couvrir tous les services ou seulement quelques services pour permettre une opération telle que le routage basé sur les versions.



Avantages

Les modifications de configuration nécessitent un minimum d'efforts et peuvent être automatisées facilement. Cette méthode est également flexible et permet de créer des moyens créatifs pour n'exposer que les opérations spécifiques que vous souhaitez obtenir d'un service.

Inconvénients

Comme pour la méthode de routage par nom d'hôte, le routage des en-têtes HTTP suppose que vous avez un contrôle total sur le client et que vous pouvez manipuler des en-têtes HTTP personnalisés. Les proxys, les réseaux de diffusion de contenu (CDN) et les équilibreurs de charge peuvent limiter la taille de l'en-tête. Bien que cela ne soit pas un problème, cela peut poser problème en fonction du nombre d'en-têtes et de cookies que vous ajoutez.

Schéma du disjoncteur

Intention

Le schéma du disjoncteur peut empêcher un service appelant de retenter un appel vers un autre service (appelé) lorsque l'appel a déjà provoqué des délais ou des échecs répétés. Le modèle est également utilisé pour détecter le moment où le service appelé est à nouveau fonctionnel.

Motivation

Lorsque plusieurs microservices collaborent pour traiter les demandes, un ou plusieurs services peuvent devenir indisponibles ou présenter une latence élevée. Lorsque des applications complexes utilisent des microservices, une panne d'un microservice peut entraîner la défaillance de l'application. Les microservices communiquent par le biais d'appels de procédure à distance, et des erreurs transitoires peuvent survenir dans la connectivité réseau et provoquer des défaillances. (Les erreurs transitoires peuvent être gérées en utilisant le modèle de nouvelle tentative avec retrait.) Lors d'une exécution synchrone, la cascade de délais d'attente ou d'échecs peut nuire à l'expérience utilisateur.

Cependant, dans certains cas, la résolution des défaillances peut prendre plus de temps, par exemple lorsque le service appelé est en panne ou qu'un conflit de base de données entraîne des délais d'attente. Dans de tels cas, si le service d'appel tente les appels à plusieurs reprises, ces tentatives peuvent entraîner un conflit sur le réseau et une consommation du pool de threads de base de données. En outre, si plusieurs utilisateurs réessayent l'application à plusieurs reprises, cela ne fera qu'aggraver le problème et entraîner une dégradation des performances de l'ensemble de l'application.

Le schéma du disjoncteur a été popularisé par Michael Nygard dans son livre, Release It (Nygard 2018). Ce modèle de conception peut empêcher un service appelant de réessayer un appel de service qui a déjà provoqué des délais ou des échecs répétés. Il peut également détecter le moment où le service appelé est à nouveau fonctionnel.

Les objets disjoncteurs fonctionnent comme des disjoncteurs électriques qui interrompent automatiquement le courant en cas d'anomalie dans le circuit. Les disjoncteurs électriques coupent ou déclenchent la circulation du courant en cas de panne. De même, l'objet disjoncteur est situé entre l'appelant et le service appelé, et se déclenche si l'appelé n'est pas disponible.

Les erreurs de l'informatique distribuée sont un ensemble d'assertions faites par Peter Deutsch et d'autres collaborateurs de Sun Microsystems. Ils disent que les programmeurs novices dans le

domaine des applications distribuées font invariablement de fausses suppositions. La fiabilité du réseau, les attentes en matière de latence nulle et les limites de bande passante se traduisent par des applications logicielles conçues avec une gestion minimale des erreurs réseau.

Lors d'une panne de réseau, les applications peuvent attendre indéfiniment une réponse et consommer continuellement les ressources des applications. Le fait de ne pas recommencer les opérations lorsque le réseau devient disponible peut également entraîner une dégradation de l'application. Si les appels d'API à une base de données ou à un service externe arrivent à expiration en raison de problèmes réseau, des appels répétés sans disjoncteur peuvent affecter les coûts et les performances.

Applicabilité

Utilisez ce modèle lorsque :

- Le service d'appel passe un appel qui risque fort d'échouer.
- Une latence élevée du service appelé (par exemple, lorsque les connexions à la base de données sont lentes) entraîne des délais d'attente pour le service appelé.
- Le service d'appel effectue un appel synchrone, mais le service appelé n'est pas disponible ou présente une latence élevée.

Problèmes et considérations

- Implémentation indépendante des services : pour éviter toute surcharge de code, nous vous recommandons d'implémenter l'objet disjoncteur d'une manière indépendante des microservices et pilotée par une API.
- Fermeture du circuit par l'appelé : lorsque l'appelé se remet d'un problème ou d'une panne de performance, il peut mettre à jour l'état du circuit sur. `CLOSED` Il s'agit d'une extension du schéma du disjoncteur qui peut être mise en œuvre si votre objectif de temps de rétablissement (RTO) l'exige.
- Appels multithread : la valeur du délai d'expiration est définie comme la période pendant laquelle le circuit reste déclenché avant que les appels ne soient à nouveau routés pour vérifier la disponibilité du service. Lorsque le service appelé est appelé dans plusieurs threads, le premier appel qui a échoué définit la valeur du délai d'expiration. Votre implémentation doit garantir que les appels suivants ne déplacent pas le délai d'expiration indéfiniment.

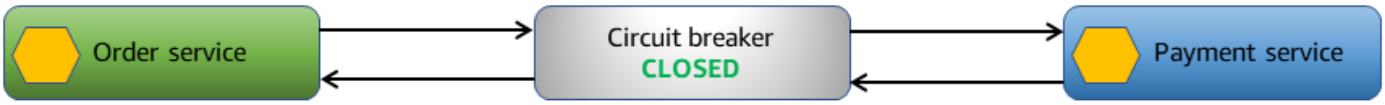
- Forcer l'ouverture ou la fermeture du circuit : les administrateurs système doivent avoir la possibilité d'ouvrir ou de fermer un circuit. Cela peut être fait en mettant à jour la valeur du délai d'expiration dans la table de base de données.
- Observabilité : L'application doit avoir une journalisation configurée pour identifier les appels qui échouent lorsque le disjoncteur est ouvert.

Mise en œuvre

Architecture de haut niveau

Dans l'exemple suivant, l'appelant est le service de commande et l'appelé est le service de paiement.

Lorsqu'il n'y a pas de panne, le service de commande achemine tous les appels vers le service de paiement par le disjoncteur, comme le montre le schéma suivant.



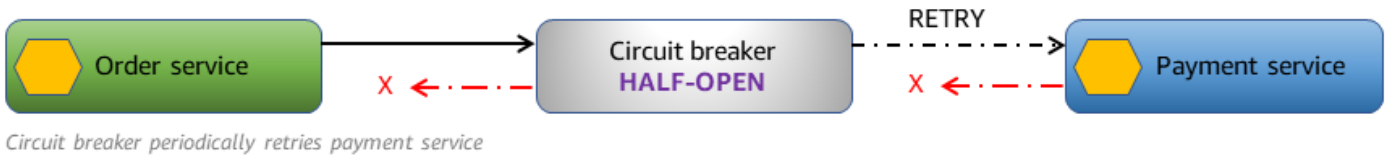
Si le délai d'expiration du service de paiement est dépassé, le disjoncteur peut détecter le délai d'expiration et suivre la panne.

Circuit breaker with payment service failure

Si les délais d'attente dépassent un seuil spécifié, l'application ouvre le circuit. Lorsque le circuit est ouvert, l'objet du disjoncteur n'achemine pas les appels vers le service de paiement. Il renvoie une panne immédiate lorsque le service de commande appelle le service de paiement.

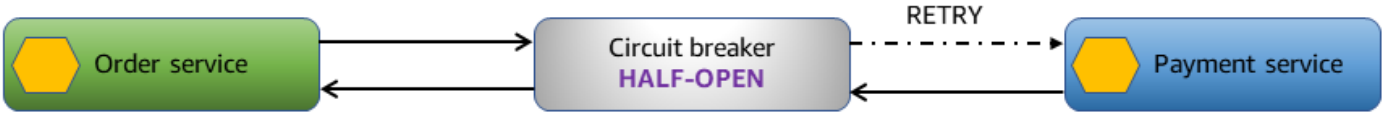
Circuit breaker stops routing to payment service

L'objet disjoncteur essaie périodiquement de voir si les appels au service de paiement aboutissent.



Circuit breaker periodically retries payment service

Lorsque l'appel au service de paiement aboutit, le circuit est fermé et tous les autres appels sont à nouveau acheminés vers le service de paiement.



Circuit breaker with working payment service

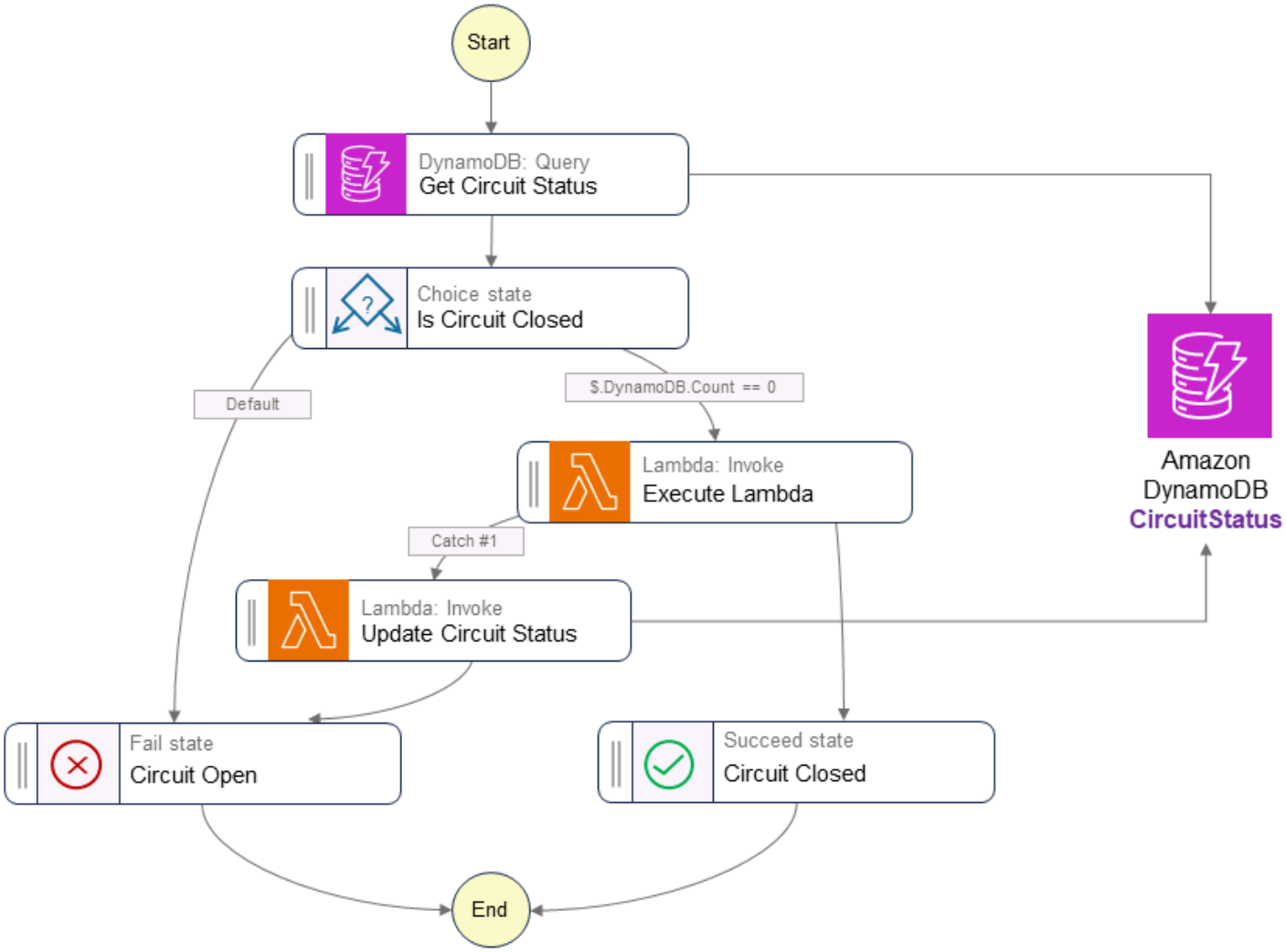
Mise en œuvre à l'aide des services AWS

L'exemple de solution utilise des flux de travail express AWS Step Functionspour implémenter le schéma du disjoncteur. La machine d'état Step Functions vous permet de configurer les capacités de nouvelle tentative et le flux de contrôle basé sur les décisions nécessaires à la mise en œuvre du modèle.

La solution utilise également une table Amazon DynamoDB comme magasin de données pour suivre l'état du circuit. Cela peut être remplacé par une banque de données en mémoire telle qu'Amazon ElastiCache pour Redis pour de meilleures performances.

Lorsqu'un service souhaite appeler un autre service, il lance le flux de travail avec le nom du service appelé. Le flux de travail obtient l'état du disjoncteur à partir de la table `CircuitStatus` DynamoDB, qui stocke les services actuellement dégradés. S'il `CircuitStatus` contient un dossier non expiré pour l'appelé, le circuit est ouvert. Le flux de travail Step Functions renvoie une défaillance immédiate et s'en sort avec un `FAIL` état.

Si la `CircuitStatus` table ne contient aucun enregistrement pour l'appelé ou contient un enregistrement expiré, le service est opérationnel. L'`ExecuteLambda`étape de définition de la machine à états appelle la fonction Lambda qui est envoyée via une valeur de paramètre. Si l'appel aboutit, le flux de travail Step Functions se termine avec un `SUCCESS` état.



En cas d'échec de l'appel de service ou d'expiration du délai imparti, l'application réessaie avec un retard exponentiel pendant un nombre défini de fois. Si l'appel de service échoue après les nouvelles tentatives, le flux de travail insère un enregistrement dans le `CircuitStatus` tableau pour le service avec un an`ExpiryTimeStamp`, et le flux de travail sort avec un `FAIL` état. Les appels ultérieurs au même service entraînent une panne immédiate tant que le disjoncteur est ouvert. L'`Get Circuit Status`étape de définition de la machine à états vérifie la disponibilité du service en fonction de la `ExpiryTimeStamp` valeur. Les éléments expirés sont supprimés de la `CircuitStatus` table à l'aide de la fonctionnalité DynamoDB Time to Live (TTL).

Exemple de code

Le code suivant utilise la fonction `GetCircuitStatus` Lambda pour vérifier l'état du disjoncteur.

```
var serviceDetails = _dbContext.QueryAsync<CircuitBreaker>(serviceName,
 QueryOperator.GreaterThan,
                new List<object>
                    {currentTimeStamp}).GetRemainingAsync();

if (serviceDetails.Result.Count > 0)
{
    functionData.CircuitStatus = serviceDetails.Result[0].CircuitStatus;
}
else
{
    functionData.CircuitStatus = "";
}
```

Le code suivant montre les instructions Amazon States Language dans le flux de travail Step Functions.

```
"Is Circuit Closed": {
    "Type": "Choice",
    "Choices": [
    {
        "Variable": "$.CircuitStatus",
        "StringEquals": "OPEN",
        "Next": "Circuit Open"
    },
    {
        "Variable": "$.CircuitStatus",
        "StringEquals": "",
        "Next": "Execute Lambda"
    }
    ]
},
"Circuit Open": {
    "Type": "Fail"
}
```

GitHub référentiel

Pour une implémentation complète de l'exemple d'architecture pour ce modèle, consultez le GitHub référentiel à l'adresse https://github.com/aws-samples/ circuit-breaker-netcore-blog.

Références du blog

- Utilisation du schéma du disjoncteur avec AWS Step Functions Amazon DynamoDB

Contenu connexe

- Motif Strangler Fig
- Retry with backoff pattern
- AWS App Meshcapacités des disjoncteurs

Modèle d’approvisionnement d’événement

Intention

Dans les architectures pilotées par les événements, le modèle d’approvisionnement d’événement stocke les événements qui entraînent un changement d’état dans un magasin de données. Cela permet de saisir et de conserver un historique complet des changements d’état, et de promouvoir l’auditabilité, la traçabilité et la capacité d’analyser les états passés.

Motivation

Plusieurs microservices peuvent collaborer pour traiter les demandes, et ils communiquent par le biais d’événements. Ces événements peuvent entraîner un changement d’état (données). Le stockage des objets d’événements dans l’ordre dans lequel ils apparaissent fournit des informations précieuses sur l’état actuel de l’entité de données et des informations supplémentaires sur la façon dont elle est arrivée à cet état.

Applicabilité

Utilisez le modèle d’approvisionnement d’événement lorsque :

- Un historique immuable des événements qui se produisent dans une application est nécessaire pour le suivi.
- Les projections de données polyglottes sont requises à partir d’une source unique de vérité (SSOT).
- Une reconstruction ponctuelle de l’état de l’application est nécessaire.
- Le stockage à long terme de l’état de l’application n’est pas nécessaire, mais que vous souhaitez peut-être le reconstruire selon vos besoins.
- Les charges de travail ont des volumes de lecture et d’écriture différents. Par exemple, vous avez des charges de travail intensives en écriture qui ne nécessitent pas de traitement en temps réel.
- La capture des données de modification (CDC) est nécessaire pour analyser les performances de l’application et d’autres métriques.
- Les données d’audit sont requises pour tous les événements survenant dans un système à des fins de génération de rapports et de conformité.

- Vous souhaitez obtenir des scénarios hypothétiques en modifiant (insertion, mise à jour ou suppression) des événements au cours du processus de rediffusion afin de déterminer l'état final possible.

Problèmes et considérations

- Contrôle de simultanéité optimiste : ce modèle stocke tous les événements qui entraînent un changement d'état dans le système. Plusieurs utilisateurs ou services peuvent essayer de mettre à jour les mêmes données en même temps, ce qui peut provoquer des collisions d'événements. Ces collisions se produisent lorsque des événements contradictoires sont créés et appliqués en même temps, ce qui se traduit par un état final des données qui ne correspond pas à la réalité. Pour résoudre ce problème, vous pouvez mettre en œuvre des stratégies pour détecter et résoudre les collisions d'événements. Par exemple, vous pouvez implémenter un schéma de contrôle de simultanéité optimiste en incluant la gestion des versions ou en ajoutant des horodatages aux événements pour suivre l'ordre des mises à jour.
- Complexité : la mise en œuvre de l'approvisionnement d'événement nécessite un changement d'état d'esprit, passant des opérations CRUD traditionnelles à une réflexion axée sur les événements. Le processus de rediffusion, qui est utilisé pour restaurer le système dans son état d'origine, peut être complexe afin de garantir l'idempotence des données. Le stockage d'événements, les sauvegardes et les instantanés peuvent également ajouter de la complexité.
- Cohérence à terme : les projections de données dérivées des événements sont cohérentes à terme en raison de la latence dans la mise à jour des données en utilisant le modèle de séparation des responsabilités des requêtes de commande (CQRS) ou des vues matérialisées. Lorsque les consommateurs traitent les données d'un magasin d'événements et que les diffuseurs de publication envoient de nouvelles données, la projection des données ou l'objet de l'application peuvent ne pas représenter l'état actuel.
- Interrogation : la récupération des données actuelles ou agrégées à partir des journaux d'événements peut être plus complexe et plus lente que celle des bases de données traditionnelles, en particulier pour les requêtes complexes et les tâches de génération de rapports. Pour atténuer ce problème, l'approvisionnement d'événement est souvent mis en œuvre selon le modèle CQRS.
- Taille et coût du magasin d'événements : le magasin d'événements peut connaître une croissance exponentielle en raison de la persistance des événements, en particulier dans les systèmes présentant un débit d'événements élevé ou des périodes de rétention prolongées. Par conséquent,

vous devez régulièrement archiver les données des événements pour les stocker de manière rentable afin d'éviter que le magasin d'événements ne devienne trop volumineux.

- Capacité de mise à l'échelle du magasin d'événements : le magasin d'événements doit gérer efficacement de gros volumes d'opérations d'écriture et de lecture. La mise à l'échelle d'un magasin d'événements peut s'avérer difficile. Il est donc important de disposer d'un magasin de données qui fournit des partitions.
- Efficacité et optimisation : choisissez ou concevez un magasin d'événements qui gère efficacement les opérations d'écriture et de lecture. Le magasin d'événements doit être optimisé en fonction du volume d'événements attendu et des modèles de requêtes pour l'application. La mise en œuvre de mécanismes d'indexation et de requête peut accélérer la récupération des événements lors de la reconstruction de l'état de l'application. Vous pouvez également envisager d'utiliser des bases de données ou des bibliothèques de magasins d'événements spécialisées qui offrent des fonctionnalités d'optimisation des requêtes.
- Instantanés : vous devez sauvegarder les journaux d'événements à intervalles réguliers avec une activation basée sur le temps. La répétition des événements survenus lors de la dernière sauvegarde réussie connue des données devrait permettre de rétablir l'état de l'application à un moment donné. L'objectif de point de reprise (RPO) est le délai maximal acceptable depuis le dernier point de reprise des données. Le RPO détermine ce qui est considéré comme une perte de données acceptable entre le dernier point de reprise et l'interruption du service. La fréquence des instantanés quotidiens du magasin de données et d'événements doit être basée sur le RPO de l'application.
- Sensibilité temporelle : les événements sont stockés dans l'ordre dans lequel ils se produisent. Par conséquent, la fiabilité du réseau est un facteur important à prendre en compte lorsque vous implémentez ce modèle. Les problèmes de latence peuvent entraîner un état incorrect du système. Utilisez des files d'attente « premier entré, premier sorti » (FIFO) avec livraison en une seule fois pour transporter les événements jusqu'au magasin qui leur est dédié.
- Performances de rediffusion d'événements : la répétition d'un grand nombre d'événements pour reconstituer l'état actuel de l'application peut prendre beaucoup de temps. Des efforts d'optimisation sont nécessaires pour améliorer les performances, en particulier lors de la rediffusion d'événements à partir de données archivées.
- Mises à jour externes du système : les applications qui utilisent le modèle d'approvisionnement d'événement peuvent mettre à jour les magasins de données dans des systèmes externes et peuvent capturer ces mises à jour sous forme d'objets d'événements. Lors des rediffusions d'événements, cela peut devenir un problème si le système externe ne s'attend pas à une mise

à jour. Dans de tels cas, vous pouvez utiliser des indicateurs de fonctionnalité pour contrôler les mises à jour externes du système.

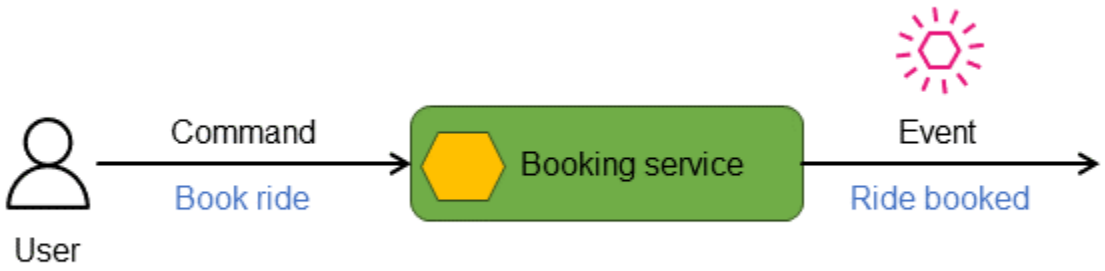
- Requêtes système externes : lorsque les appels système externes sont sensibles à la date et à l'heure de l'appel, les données reçues peuvent être stockées dans des magasins de données internes pour être utilisées lors des rediffusions.
- Gestion des versions des événements : au fur et à mesure que l'application évolue, la structure des événements (schéma) peut changer. La mise en œuvre d'une stratégie de gestion des versions pour les événements afin de garantir la rétrocompatibilité et la compatibilité descendante est nécessaire. Cela peut impliquer l'inclusion d'un champ de version dans la charge utile de l'événement et la gestion des différentes versions de l'événement de manière appropriée pendant la rediffusion.

Mise en œuvre

Architecture de haut niveau

Commandes et événements

Dans les applications de microservices distribuées pilotées par des événements, les commandes représentent les instructions ou les demandes envoyées à un service, généralement dans le but de modifier son état. Le service traite ces commandes et évalue leur validité et leur applicabilité à son état actuel. Si la commande s'exécute correctement, le service répond en émettant un événement indiquant l'action entreprise et les informations d'état pertinentes. Par exemple, dans le schéma suivant, le service de réservation répond à la commande Réserver un trajet en émettant l'événement Trajet réservé.



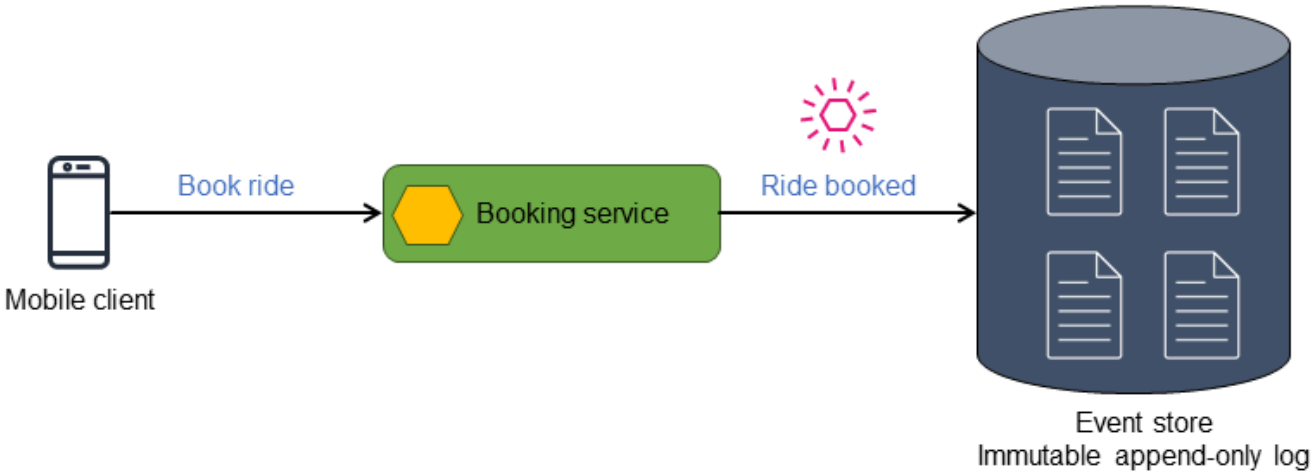
Magasins d'événements

Les événements sont journalisés dans un référentiel ou un magasin de données immuable, avec ajout uniquement et ordonné chronologiquement, appelé magasin d'événements. Chaque changement d'état est traité comme un objet d'événement individuel. Un objet entité ou un magasin de données dont l'état initial est connu, son état actuel et n'importe quelle vue ponctuelle peuvent être reconstruits en rejouant les événements dans l'ordre de leur apparition.

Le magasin d'événements sert de registre historique de toutes les actions et de tous les changements d'état, et constitue une source unique et précieuse de vérité. Vous pouvez utiliser le magasin d'événements pour obtenir l'état final à jour du système en transmettant les événements à un processeur de rediffusion, qui applique ces événements pour produire une représentation précise de l'état le plus récent du système. Vous pouvez également utiliser le magasin d'événements pour générer la perspective ponctuelle de l'état en rejouant les événements via un processeur de rediffusion. Dans le modèle d'approvisionnement d'événement, l'état actuel peut ne pas être entièrement représenté par l'objet d'événement le plus récent. Vous pouvez obtenir l'état actuel de l'une des trois façons suivantes :

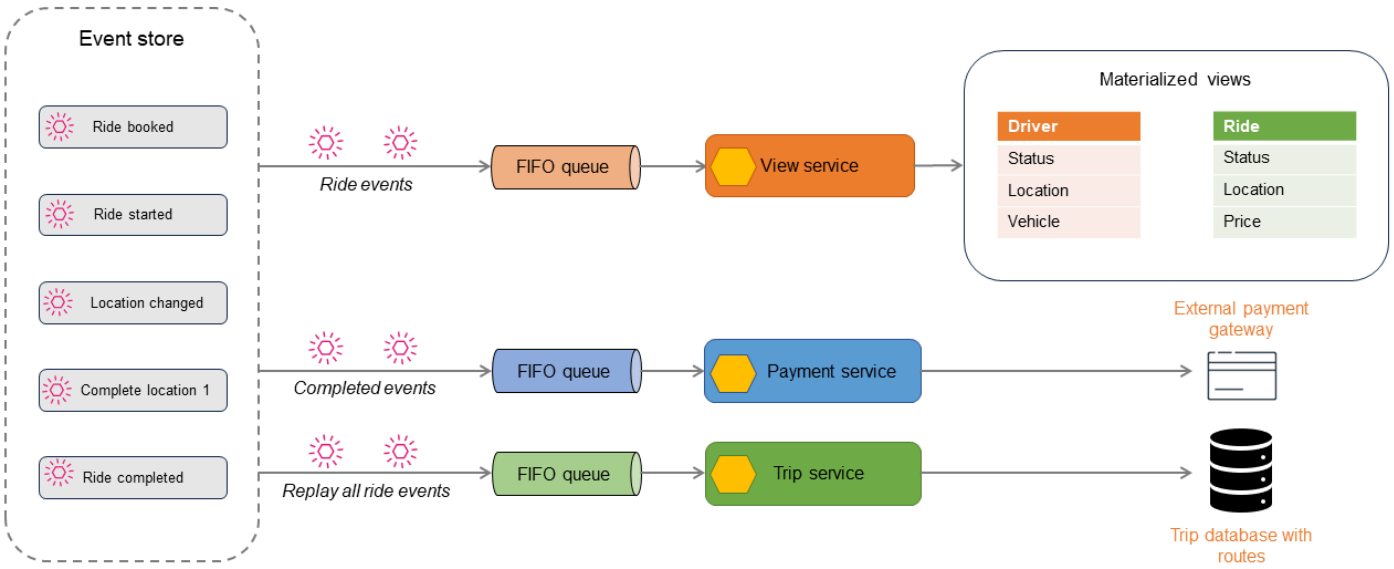
- En agrégeant les événements associés. Les objets d'événements associés sont combinés pour générer l'état actuel pour les requêtes. Cette approche est souvent utilisée conjointement avec le modèle CQRS, dans lequel les événements sont combinés et écrits dans le magasin de données en lecture seule.
- En utilisant les vues matérialisées. Vous pouvez utiliser l'approvisionnement d'événement avec le modèle de vue matérialisée pour calculer ou résumer les données d'événements et obtenir l'état actuel des données associées.
- En rejouant les événements. Les objets d'événement peuvent être rejoués pour effectuer des actions permettant de générer l'état actuel.

Le schéma suivant montre l'événement `Ride booked` stocké dans un magasin d'événements.



Le magasin d'événements publie les événements qu'il stocke, et les événements peuvent être filtrés et acheminés vers le processeur approprié pour les actions suivantes. Par exemple, les événements peuvent être acheminés vers un processeur de vues qui résume l'état et affiche une vue matérialisée. Les événements sont transformés au format de données du magasin de données cible. Cette architecture peut être étendue pour dériver différents types de magasins de données, ce qui entraîne une persistance polyglotte des données.

Le schéma suivant décrit les événements d'une application de réservation de trajets. Tous les événements qui se produisent dans l'application sont stockés dans le magasin d'événements. Les événements stockés sont ensuite filtrés et acheminés vers différents utilisateurs.



Les événements de trajet peuvent être utilisés pour générer des magasins de données en lecture seule à l'aide du modèle CQRS ou du modèle de vue matérialisée. Vous pouvez obtenir l'état actuel du trajet, du chauffeur ou de la réservation en interrogeant les magasins de lecture. Certains événements, tels que `Location changed` ou `Ride completed`, sont communiqués à un autre utilisateur pour le traitement des paiements. Une fois le trajet terminé, tous les événements du trajet sont rejoués pour créer un historique du trajet à des fins d'audit ou de génération de rapports.

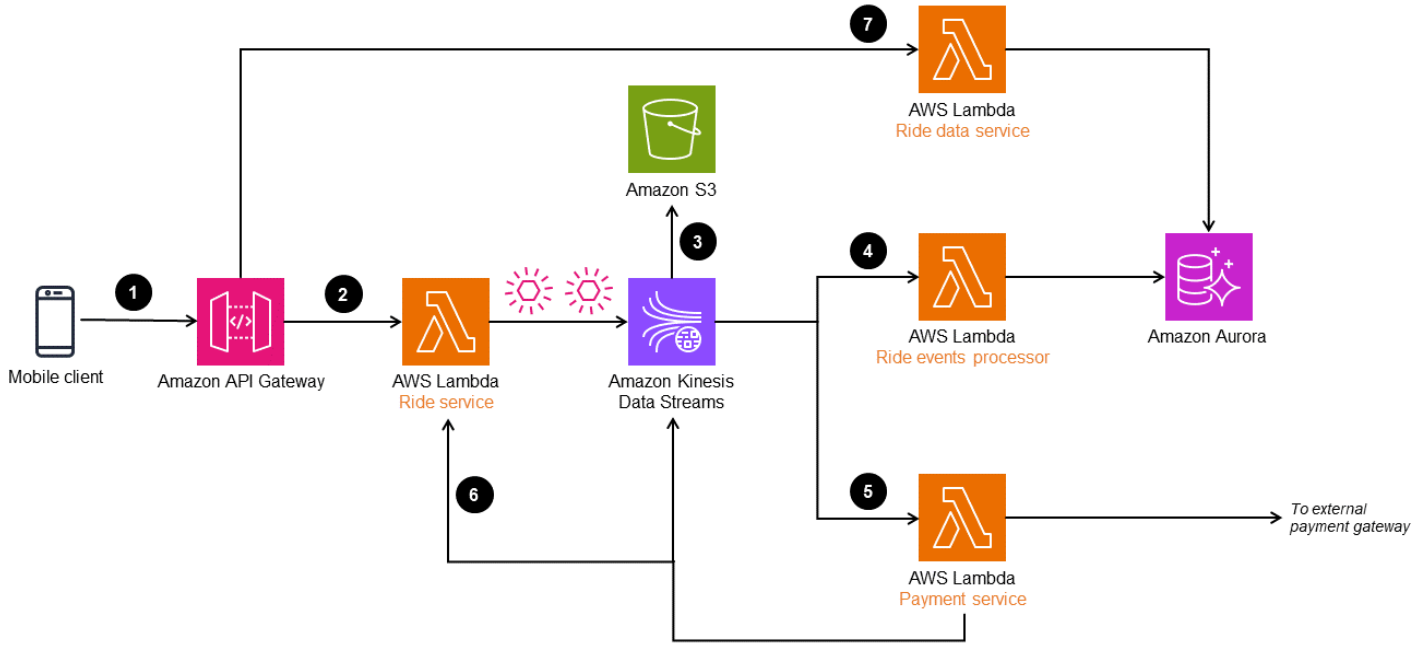
Le modèle d'approvisionnement d'événement est fréquemment utilisé dans les applications qui nécessitent une récupération ponctuelle, ainsi que lorsque les données doivent être projetées dans différents formats à l'aide d'une source unique de vérité. Ces deux opérations nécessitent un processus de rediffusion pour exécuter les événements et obtenir l'état final requis. Le processeur de rediffusion peut également avoir besoin d'un point de départ connu, idéalement pas dès le lancement de l'application, car ce ne serait pas un processus efficace. Nous vous recommandons de prendre régulièrement des instantanés de l'état du système et d'appliquer un plus petit nombre d'événements pour obtenir un état actualisé.

Mise en œuvre à l'aide des services AWS

Dans l'architecture suivante, Amazon Kinesis Data Streams est utilisé comme magasin d'événements. Ce service capture et gère les modifications des applications sous forme d'événements, et offre une solution de diffusion de flux de données à haut débit et en temps réel. Pour implémenter le modèle d'approvisionnement d'événement sur AWS, vous pouvez également

utiliser des services tels qu'Amazon EventBridge et Amazon Managed Streaming for Apache Kafka (Amazon MSK) en fonction des besoins de votre application.

Pour améliorer la durabilité et activer l'audit, vous pouvez archiver les événements capturés par Kinesis Data Streams dans Amazon Simple Storage Service (Amazon S3). Cette méthode à double stockage permet de retenir les données historiques des événements en toute sécurité pour les analyses futures et à des fins de conformité.



Le flux de travail se compose des étapes suivantes :

1. Une demande de réservation de trajet est envoyée via un client mobile à un point de terminaison Amazon API Gateway.
2. Le microservice de trajet (fonction Lambda `Ride service`) reçoit la demande, transforme les objets et publie sur Kinesis Data Streams.
3. Les données d'événements contenues dans Kinesis Data Streams sont stockées dans Amazon S3 à des fins de conformité et d'historique des audits.
4. Les événements sont transformés et traités par la fonction Lambda `Ride event processor` et stockés dans une base de données Amazon Aurora afin de fournir une vue matérialisée des données de trajet.
5. Les événements de trajet terminés sont filtrés et envoyés pour traitement du paiement à une passerelle de paiement externe. Lorsque le paiement est effectué, un autre événement est envoyé à Kinesis Data Streams pour mettre à jour la base de données de trajets.

6. Une fois le trajet terminé, les événements de trajet sont rejoués dans la fonction Lambda `Ride service` pour créer des itinéraires et l'historique du trajet.
7. Les informations sur les trajets peuvent être lues via le `Ride data service`, qui se lit dans la base de données Aurora.

API Gateway peut également envoyer l'objet d'événement directement à Kinesis Data Streams sans la fonction Lambda `Ride service`. Toutefois, dans un système complexe tel qu'un service de transport, l'objet de l'événement peut avoir besoin d'être traité et enrichi avant d'être ingéré dans le flux de données. C'est pourquoi l'architecture dispose d'un `Ride service` qui traite l'événement avant de l'envoyer à Kinesis Data Streams.

Références du blog

- New for AWS Lambda – SQS FIFO as an event source

Motif d'architecture hexagonal

Intention

Le modèle d'architecture hexagonal, également connu sous le nom de modèle de ports et d'adaptateurs, a été proposé par le Dr Alistair Cockburn en 2005. Il vise à créer des architectures faiblement couplées dans lesquelles les composants de l'application peuvent être testés indépendamment, sans dépendance vis-à-vis des magasins de données ou des interfaces utilisateur (UI). Ce modèle permet d'éviter le verrouillage technologique des banques de données et des interfaces utilisateur. Cela facilite l'évolution de la pile technologique au fil du temps, avec un impact limité ou nul sur la logique métier. Dans cette architecture faiblement couplée, l'application communique avec les composants externes via des interfaces appelées ports, et utilise des adaptateurs pour traduire les échanges techniques avec ces composants.

Motivation

Le modèle d'architecture hexagonal est utilisé pour isoler la logique métier (logique de domaine) du code d'infrastructure associé, tel que le code permettant d'accéder à une base de données ou à des API externes. Ce modèle est utile pour créer une logique métier et un code d'infrastructure faiblement couplés pour les AWS Lambda fonctions nécessitant une intégration avec des services externes. Dans les architectures traditionnelles, une pratique courante consiste à intégrer la logique métier dans la couche de base de données sous forme de procédures stockées et dans l'interface utilisateur. Cette pratique, associée à l'utilisation de structures spécifiques à l'interface utilisateur dans la logique métier, conduit à des architectures étroitement couplées qui entravent les migrations de bases de données et les efforts de modernisation de l'expérience utilisateur (UX). Le modèle d'architecture hexagonal vous permet de concevoir vos systèmes et applications par objectif plutôt que par technologie. Cette stratégie permet de créer des composants d'application facilement échangeables tels que les bases de données, l'expérience utilisateur et les composants de service.

Applicabilité

Utilisez le modèle d'architecture hexagonal lorsque :

- Vous souhaitez dissocier l'architecture de votre application afin de créer des composants pouvant être entièrement testés.
- Plusieurs types de clients peuvent utiliser la même logique de domaine.

- Les composants de votre interface utilisateur et de votre base de données nécessitent des mises à jour technologiques périodiques qui n'affectent pas la logique de l'application.
- Votre application nécessite plusieurs fournisseurs d'entrées et consommateurs de sortie, et la personnalisation de la logique de l'application entraîne une complexité du code et un manque d'extensibilité.

Problèmes et considérations

- Conception axée sur le domaine : L'architecture hexagonale fonctionne particulièrement bien avec la conception axée sur le domaine (DDD). Chaque composant d'application représente un sous-domaine dans DDD, et les architectures hexagonales peuvent être utilisées pour réaliser un couplage souple entre les composants de l'application.
- Testabilité : De par sa conception, une architecture hexagonale utilise des abstractions pour les entrées et les sorties. Par conséquent, l'écriture de tests unitaires et de tests isolés devient plus facile en raison du couplage lâche inhérent.
- Complexité : la complexité liée à la séparation de la logique métier du code d'infrastructure, lorsqu'elle est gérée avec soin, peut apporter de grands avantages tels que l'agilité, la couverture des tests et l'adaptabilité technologique. Sinon, les problèmes peuvent devenir complexes à résoudre.
- Frais de maintenance : le code d'adaptateur supplémentaire qui rend l'architecture enfichable n'est justifié que si le composant de l'application nécessite plusieurs sources d'entrée et destinations de sortie sur lesquelles écrire, ou lorsque le magasin de données d'entrée et de sortie doit changer au fil du temps. Dans le cas contraire, l'adaptateur devient une couche supplémentaire à gérer, ce qui entraîne une surcharge de maintenance.
- Problèmes de latence : l'utilisation de ports et d'adaptateurs ajoute une couche supplémentaire, ce qui peut entraîner une latence.

Mise en œuvre

Les architectures hexagonales permettent d'isoler l'application et la logique métier du code d'infrastructure et du code qui intègre l'application aux interfaces utilisateur, aux API externes, aux bases de données et aux courtiers de messages. Vous pouvez facilement connecter des composants de logique métier à d'autres composants (tels que des bases de données) de l'architecture de l'application via des ports et des adaptateurs.

Les ports sont des points d'entrée indépendants de la technologie dans un composant d'application. Ces interfaces personnalisées déterminent l'interface qui permet aux acteurs externes de communiquer avec le composant de l'application, indépendamment de la personne ou de l'élément qui implémente l'interface. Cela ressemble à la façon dont un port USB permet à de nombreux types de périphériques de communiquer avec un ordinateur, à condition qu'ils utilisent un adaptateur USB.

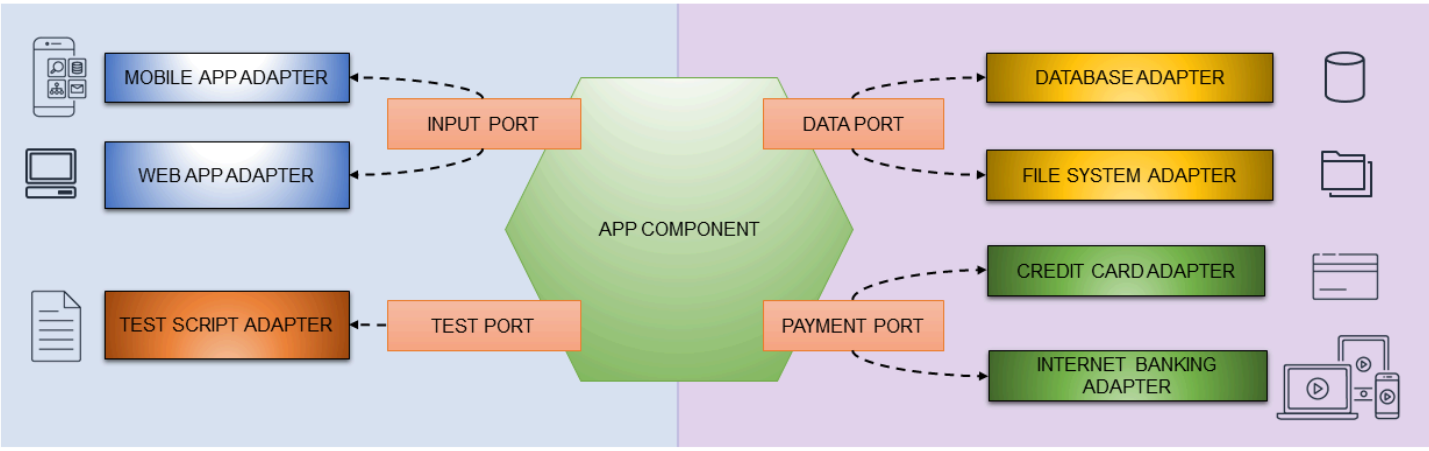
Les adaptateurs interagissent avec l'application via un port en utilisant une technologie spécifique. Les adaptateurs se connectent à ces ports, reçoivent des données depuis ou fournissent des données aux ports, et transforment les données pour un traitement ultérieur. Par exemple, un adaptateur REST permet aux acteurs de communiquer avec le composant de l'application via une API REST. Un port peut comporter plusieurs adaptateurs sans aucun risque pour le port ou le composant de l'application. Pour étendre l'exemple précédent, l'ajout d'un adaptateur GraphQL au même port fournit aux acteurs un moyen supplémentaire d'interagir avec l'application via une API GraphQL sans affecter l'API REST, le port ou l'application.

Les ports se connectent à l'application et les adaptateurs servent de connexion au monde extérieur. Vous pouvez utiliser les ports pour créer des composants d'application faiblement couplés et échanger des composants dépendants en modifiant l'adaptateur. Cela permet au composant de l'application d'interagir avec les entrées et sorties externes sans avoir besoin de connaître le contexte. Les composants sont interchangeables à tous les niveaux, ce qui facilite les tests automatisés. Vous pouvez tester les composants indépendamment sans aucune dépendance vis-à-vis du code d'infrastructure au lieu de configurer un environnement complet pour effectuer les tests. La logique de l'application ne dépend pas de facteurs externes. Les tests sont donc simplifiés et il est plus facile de simuler les dépendances.

Par exemple, dans une architecture faiblement couplée, un composant d'application doit être capable de lire et d'écrire des données sans connaître les détails du magasin de données. La responsabilité du composant d'application est de fournir des données à une interface (port). Un adaptateur définit la logique d'écriture dans un magasin de données, qui peut être une base de données, un système de fichiers ou un système de stockage d'objets tel qu'Amazon S3, en fonction des besoins de l'application.

Architecture de haut niveau

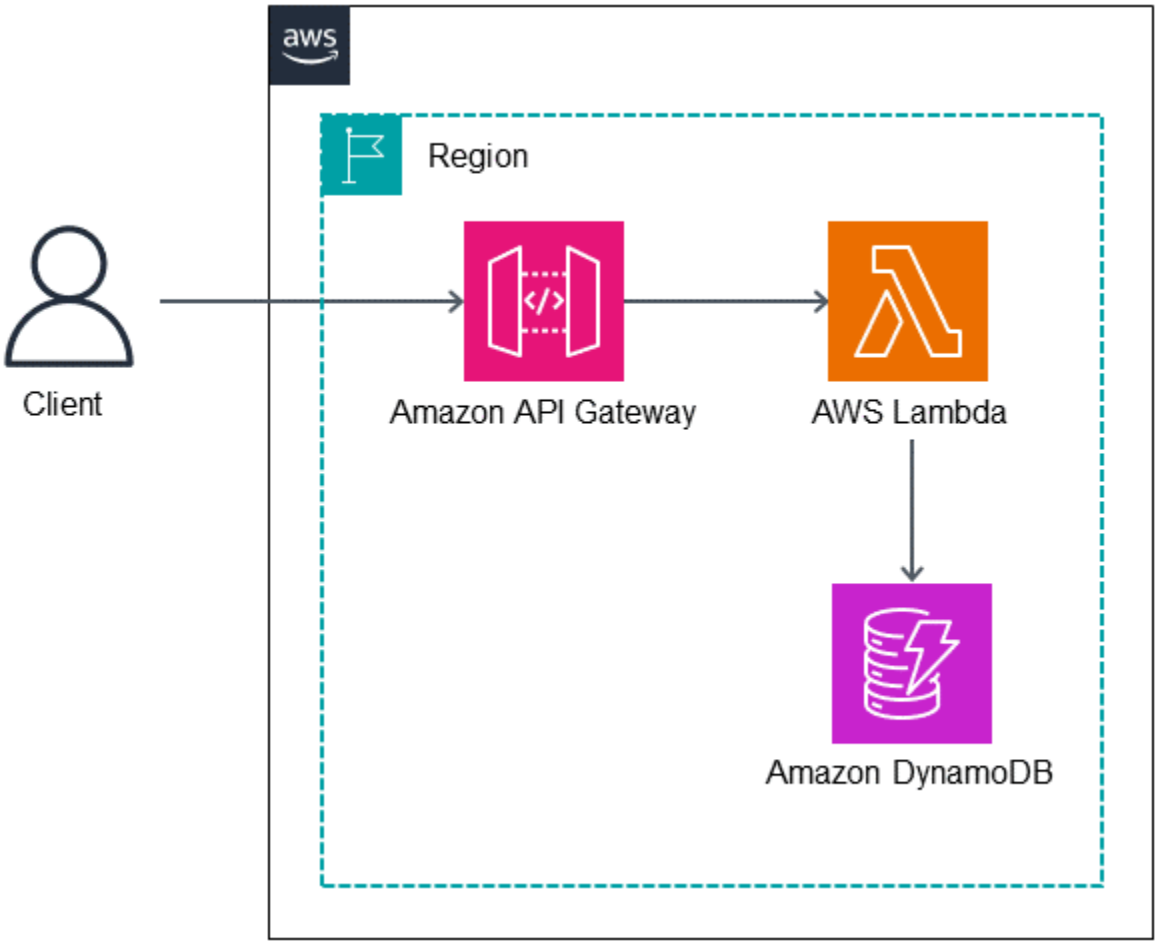
L'application ou le composant d'application contient la logique métier de base. Il reçoit des commandes ou des requêtes provenant des ports et envoie des demandes via les ports à des acteurs externes, qui sont mises en œuvre via des adaptateurs, comme illustré dans le schéma suivant.



Mise en œuvre en utilisant Services AWS

AWS Lambda les fonctions contiennent souvent à la fois une logique métier et un code d'intégration de base de données, qui sont étroitement liés pour atteindre un objectif. Vous pouvez utiliser le modèle d'architecture hexagonal pour séparer la logique métier du code d'infrastructure. Cette séparation permet des tests unitaires de la logique métier sans aucune dépendance vis-à-vis du code de base de données, et améliore l'agilité du processus de développement.

Dans l'architecture suivante, une fonction Lambda implémente le modèle d'architecture hexagonal. La fonction Lambda est initiée par l'API REST Amazon API Gateway. La fonction implémente la logique métier et écrit des données dans des tables DynamoDB.



Exemple de code

L'exemple de code présenté dans cette section montre comment implémenter le modèle de domaine à l'aide de Lambda, le séparer du code d'infrastructure (tel que le code pour accéder à DynamoDB) et implémenter des tests unitaires pour la fonction.

Modèle de domaine

La classe de modèle de domaine n'a aucune connaissance des composants ou dépendances externes : elle met uniquement en œuvre la logique métier. Dans l'exemple suivant, la classe `Recipient` est une classe de modèle de domaine qui vérifie les chevauchements dans la date de réservation.

```
class Recipient:
def init(self, recipient_id:str, email:str, first_name:str, last_name:str, age:int):
self.__recipient_id = recipient_id
self.__email = email
```

```
self.__first_name = first_name
self.__last_name = last_name
self.__age = age
self.__slots = []

@property
def recipient_id(self):
return self.__recipient_id
.....

def are_slots_same_date(self, slot:Slot) -> bool:
for selfslot in self.__slots:
if selfslot.reservation_date == slot.reservation_date:
return True
return False

def is_slot_counts_equal_or_over_two(self) -> bool:
.....
```

Port d'entrée

La `RecipientInputPort` classe se connecte à la classe destinataire et exécute la logique du domaine.

```
class RecipientInputPort(IRecipientInputPort):
def init(self, recipient_output_port: IRecipientOutputPort, slot_output_port:
 ISlotOutputPort):
self.__recipient_output_port = recipient_output_port
self.__slot_output_port = slot_output_port

def make_reservation(self, recipient_id:str, slot_id:str) -> Status:
status = None

recipient = self.__recipient_output_port.get_recipient_by_id(recipient_id)
slot = self.__slot_output_port.get_slot_by_id(slot_id)
.....

# ------------------------------------------------
# execute domain logic
# ------------------------------------------------
ret = recipient.add_reserve_slot(slot)
.....
```

```
if ret == True:
status = Status(200, "The recipient's reservation is added.")
else:
status = Status(200, "The recipient's reservation is NOT added!")
return status
```

Classe d'adaptateur DynamoDB

La `DDBRecipientAdapter` classe implémente l'accès aux tables DynamoDB.

```
class DDBRecipientAdapter(IRecipientAdapter):
def init(self):
ddb = boto3.resource('dynamodb')
self.__table = ddb.Table(table_name)

def load(self, recipient_id:str) -> Recipient:
try:
response = self.__table.get_item(
Key={'pk': pk_prefix + recipient_id})
  ...

def save(self, recipient:Recipient) -> bool:
try:
item = {
"pk": pk_prefix + recipient.recipient_id,
"email": recipient.email,
"first_name": recipient.first_name,
"last_name": recipient.last_name,
"age": recipient.age,
"slots": []
}
...
```

La fonction Lambda `get_recipient_input_port` est une fabrique pour les instances de la `RecipientInputPort` classe. Il construit des instances de classes de ports de sortie avec des instances d'adaptateur associées.

```
def get_recipient_input_port():
return RecipientInputPort(
RecipientOutputPort(DDBRecipientAdapter()),
SlotOutputPort(DDBSlotAdapter()))
```

```
def lambda_handler(event, context):

body = json.loads(event['body'])
recipient_id = body['recipient_id']
slot_id = body['slot_id']

# get an input port instance
recipient_input_port = get_recipient_input_port()
status = recipient_input_port.make_reservation(recipient_id, slot_id)

return {
"statusCode": status.status_code,
"body": json.dumps({
"message": status.message
}),
}
```

Tests d'unité

Vous pouvez tester la logique métier des classes de modèles de domaine en injectant des classes fictives. L'exemple suivant fournit le test unitaire pour la `Recipent` classe de modèle de domaine.

```
def test_add_slot_one(fixture_recipient, fixture_slot):
slot = fixture_slot
target = fixture_recipient
target.add_reserve_slot(slot)
assert slot != None
assert target != None
assert 1 == len(target.slots)
assert slot.slot_id == target.slots[0].slot_id
assert slot.reservation_date == target.slots[0].reservation_date
assert slot.location == target.slots[0].location
assert False == target.slots[0].is_vacant

def test_add_slot_two(fixture_recipient, fixture_slot, fixture_slot_2):
.....

def test_cannot_append_slot_more_than_two(fixture_recipient, fixture_slot,
 fixture_slot_2, fixture_slot_3):
.....

def test_cannot_append_same_date_slot(fixture_recipient, fixture_slot):
.....
```

GitHub référentiel

Pour une implémentation complète de l'exemple d'architecture pour ce modèle, consultez le GitHub référentiel à l'adresse https://github.com/aws-samples/ aws-lambda-domain-model -sample.

Contenu connexe

- Architecture hexagonale, article d'Alistair Cockburn
- Développement d'architectures évolutives avec AWS Lambda (article de AWS blog en japonais)

Vidéos

La vidéo suivante (en japonais) explique l'utilisation de l'architecture hexagonale dans la mise en œuvre d'un modèle de domaine à l'aide d'une fonction Lambda.

Modèle publier/s’abonner

Intention

Le modèle publier/s’abonner, également connu sous le nom de modèle pub-sub, est un modèle de messagerie qui dissocie l’expéditeur d’un message (diffuseur de publication) des destinataires intéressés (abonnés). Ce modèle implémente des communications asynchrones en publiant des messages ou des événements par le biais d’un intermédiaire appelé agent de messages ou routeur (infrastructure de messages). Le modèle publier/s’abonner renforce la capacité de mise à l’échelle et la réactivité des expéditeurs en déléguant la responsabilité de la livraison des messages à l’infrastructure de messagerie, afin que l’expéditeur puisse se concentrer sur le traitement de base des messages.

Motivation

Dans les architectures distribuées, les composants du système doivent souvent fournir des informations aux autres composants au fur et à mesure que des événements se produisent dans le système. Le modèle publier/s’abonner sépare les préoccupations afin que les applications puissent se concentrer sur leurs fonctionnalités de base, tandis que l’infrastructure de messagerie gère les responsabilités de communication telles que le routage des messages et la fiabilité de la livraison. Le modèle publier/s’abonner permet à la messagerie asynchrone de dissocier le diffuseur de publication des abonnés. Les diffuseurs de publication peuvent également envoyer des messages à l’insu des abonnés.

Applicabilité

Utilisez le modèle publier/s’abonner lorsque :

- Un traitement parallèle est nécessaire si un même message a des flux de travail différents.
- La diffusion de messages à plusieurs abonnés et les réponses en temps réel des destinataires ne sont pas nécessaires.
- Le système ou l’application peut tolérer une cohérence à terme des données ou de l’état.
- L’application ou le composant doit communiquer avec d’autres applications ou services susceptibles d’utiliser des langages, des protocoles ou des plateformes différents.

Problèmes et considérations

- Disponibilité des abonnés : le diffuseur de publication ne sait pas si les abonnés écoutent, et ce n'est peut-être pas le cas. Les messages diffusés sont de nature transitoire et peuvent être supprimés si les abonnés ne sont pas disponibles.
- Garantie de livraison des messages : en général, le modèle publier/s'abonner ne garantit pas la livraison des messages à tous les types d'abonnés, bien que certains services tels qu'Amazon Simple Notification Service (Amazon SNS) puissent fournir une livraison unique à certains sous-groupes d'abonnés.
- Durée de vie (TTL) : les messages ont une durée de vie et expirent s'ils ne sont pas traités dans le délai imparti. Envisagez d'ajouter les messages diffusés à une file d'attente afin qu'ils puissent être conservés et garantir leur traitement au-delà de la période TTL.
- Pertinence du message : les producteurs peuvent définir une période de pertinence dans le cadre des données du message, et le message peut être supprimé après cette date. Envisagez de faire en sorte que les consommateurs examinent ces informations avant de décider comment traiter le message.
- Cohérence à terme : il existe un délai entre le moment où le message est diffusé et le moment où il est consommé par l'abonné. Cela peut amener les magasins de données des abonnés à devenir cohérents à terme lorsqu'une forte cohérence est requise. La cohérence à terme peut également poser problème lorsque les producteurs et les consommateurs ont besoin d'une interaction en temps quasi réel.
- Communication unidirectionnelle : le modèle publier/s'abonner est considéré comme unidirectionnel. Les applications qui nécessitent une messagerie bidirectionnelle avec un canal d'abonnement de retour devraient envisager d'utiliser un modèle de demande de réponse si une réponse synchrone est requise.
- Ordre des messages : l'ordre des messages n'est pas garanti. Si les consommateurs ont besoin de messages ordonnés, nous vous recommandons d'utiliser les rubriques FIFO d'Amazon SNS pour garantir leur mise en ordre.
- Duplication des messages : sur la base de l'infrastructure de messagerie, des messages dupliqués peuvent être envoyés aux consommateurs. Les consommateurs doivent être conçus de manière à être idempotents pour gérer le traitement des messages dupliqués. Vous pouvez également utiliser les rubriques FIFO d'Amazon SNS pour garantir une livraison en une seule fois.
- Filtrage des messages : les consommateurs ne s'intéressent souvent qu'à un sous-ensemble des messages diffusés par un producteur. Fournissez des mécanismes permettant aux abonnés de

filtrer ou de restreindre les messages qu'ils reçoivent en fournissant des rubriques ou des filtres de contenu.

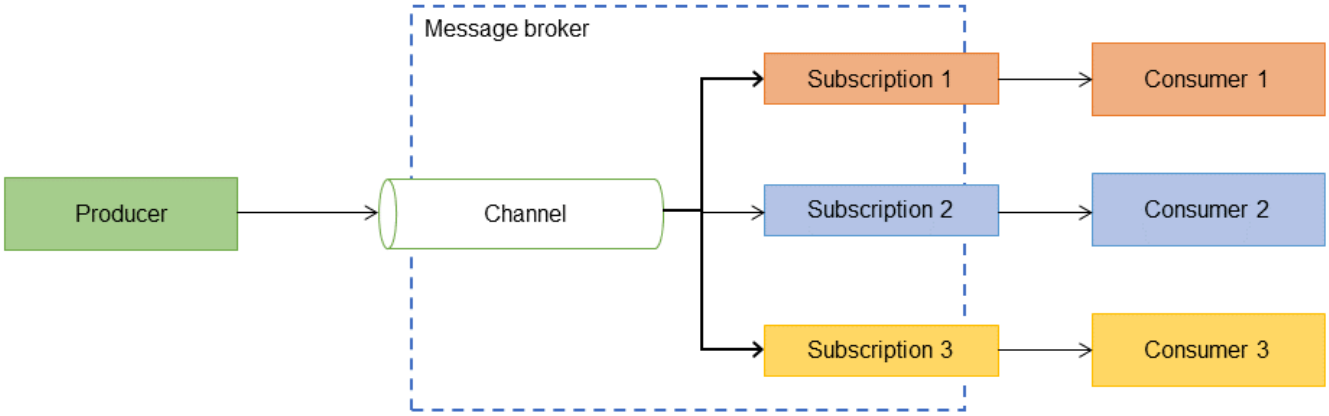
- Rediffusion des messages : les capacités de rediffusion des messages peuvent dépendre de l'infrastructure de messagerie. Vous pouvez également fournir des implémentations personnalisées en fonction du cas d'utilisation.
- Files d'attente de lettres mortes : dans un système postal, un bureau des lettres mortes est un service de traitement du courrier non distribuable. Dans la messagerie pub/sub, une file d'attente de lettres mortes (DLQ) est une file d'attente pour les messages qui ne peuvent pas être remis à un point de terminaison abonné.

Mise en œuvre

Architecture de haut niveau

Dans un modèle publier/s'abonner, le sous-système de messagerie asynchrone connu sous le nom d'agent de messages ou de routeur assure le suivi des abonnements. Lorsqu'un producteur diffuse un événement, l'infrastructure de messagerie envoie un message à chaque consommateur. Une fois qu'un message est envoyé aux abonnés, il est supprimé de l'infrastructure de messagerie afin qu'il ne puisse pas être rediffusé, et les nouveaux abonnés ne voient pas l'événement. Les agents de messages ou les routeurs dissocient le producteur d'événements des consommateurs de messages en :

- Fournissant un canal d'entrée permettant au producteur de diffuser des événements regroupés dans des messages, en utilisant un format de message défini.
- Créant un canal de sortie individuel par abonnement. Un abonnement est la connexion du consommateur, grâce à laquelle il écoute les messages d'événements associés à un canal d'entrée spécifique.
- Copiant les messages du canal d'entrée vers le canal de sortie pour tous les consommateurs lorsque l'événement est diffusé.



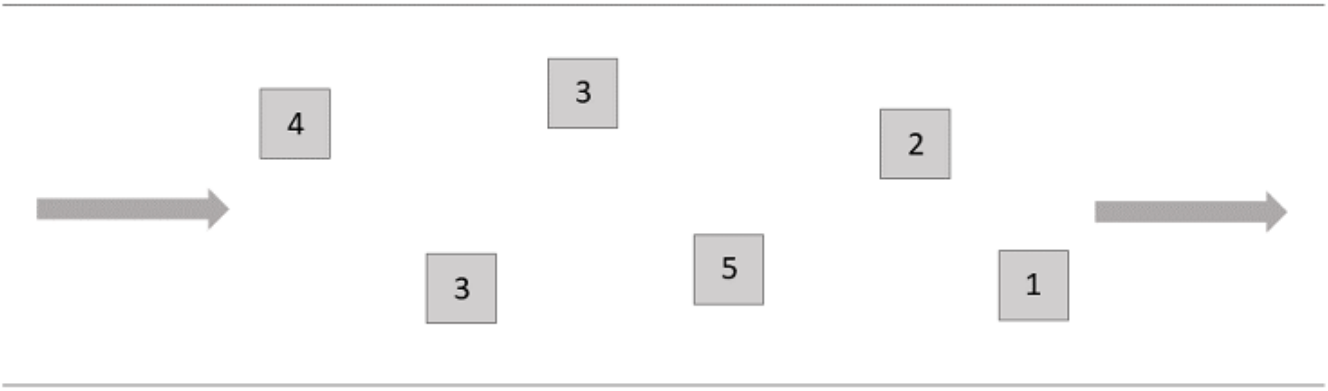
Mise en œuvre à l'aide des services AWS

Amazon SNS

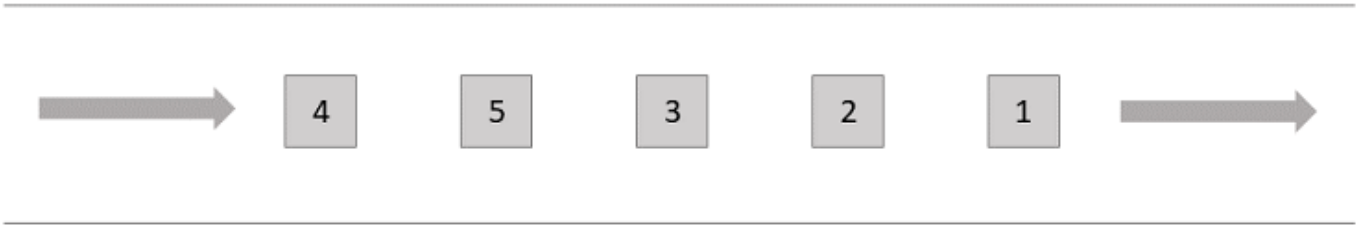
Amazon SNS est un service publier/s'abonner entièrement géré qui fournit une messagerie d'application à application (A2A) pour dissocier les applications distribuées. Il fournit également une messagerie d'application à personne (A2P) pour l'envoi de SMS, d'e-mails et d'autres notifications push.

Amazon SNS propose deux types de rubriques : standard et premier entré, premier sorti (FIFO).

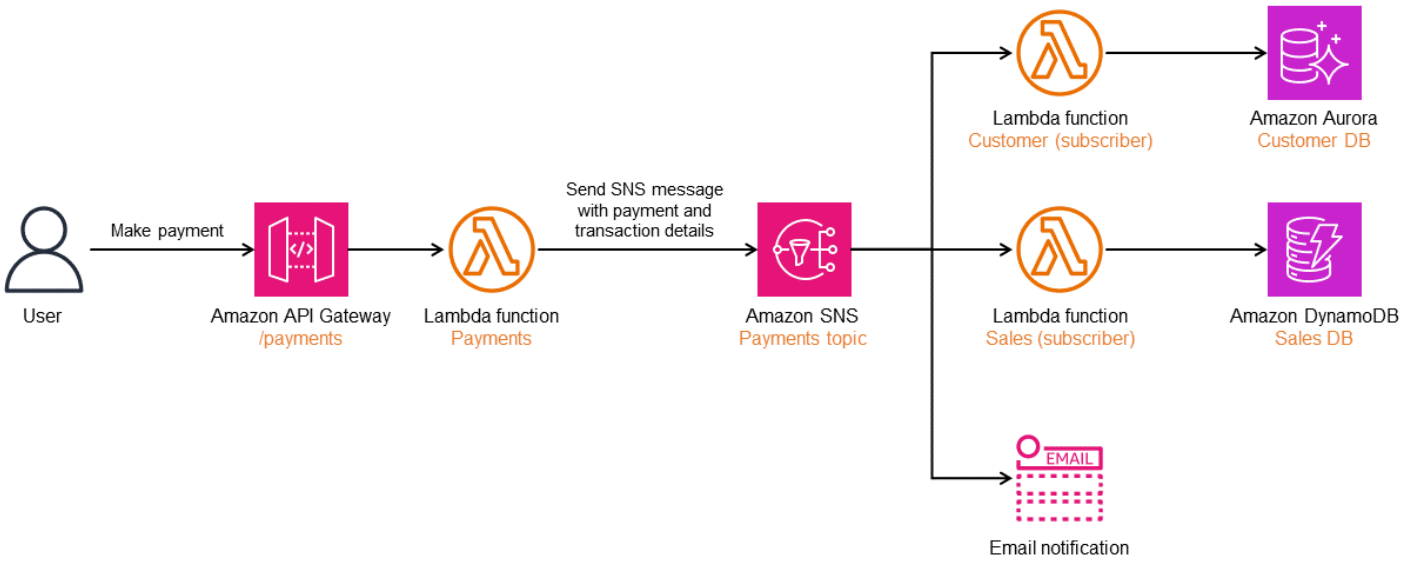
- Les rubriques standard prennent en charge un nombre illimité de messages par seconde et permettent un tri et une dissociation optimaux.



- Les rubriques FIFO fournissent un ordre et une dissociation stricts, et prennent en charge jusqu'à 300 messages par seconde ou 10 Mo par seconde par rubrique FIFO (selon la première éventualité).



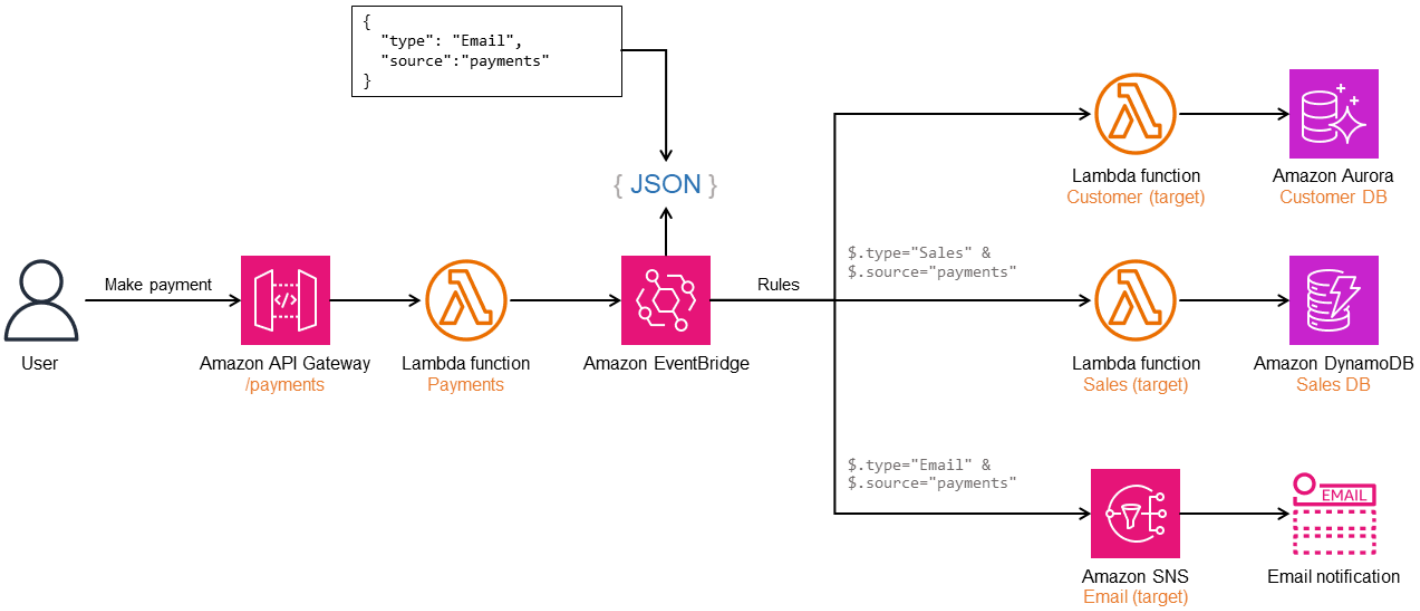
L'illustration suivante montre comment utiliser Amazon SNS pour implémenter le modèle publier/s'abonner. Une fois qu'un utilisateur a effectué un paiement, un message SNS est envoyé par la fonction Lambda `Payments` à la rubrique SNS `Payments`. Cette rubrique SNS compte trois abonnés. Chaque abonné reçoit une copie du message et le traite.



Amazon EventBridge

Vous pouvez utiliser Amazon EventBridge lorsque vous avez besoin d'un routage plus complexe des messages provenant de plusieurs producteurs via différents protocoles vers des clients abonnés, ou d'abonnements directs et en éventail. EventBridge prend également en charge le routage, le filtrage, le séquençage, le fractionnement ou l'agrégation basés sur le contenu. Dans l'illustration suivante, EventBridge est utilisé pour créer une version du modèle publier/s'abonner dans lequel les abonnés sont définis à l'aide de règles d'événement. Une fois qu'un utilisateur a effectué un paiement, la fonction Lambda `Payments` envoie un message à EventBridge en utilisant le bus d'événements par

défaut basé sur un schéma personnalisé comportant trois règles pointant vers des cibles différentes. Chaque microservice traite les messages et exécute les actions requises.



Ateliers

- Building event-driven architectures on AWS
- Diffusion en éventail de notifications d'événements avec Amazon Simple Queue Service (SQS) et Amazon Simple Notification Service (SNS)

Références du blog

- Choosing between messaging services for serverless applications
- Designing durable serverless applications with DLQs for Amazon SNS, Amazon SQS, AWS Lambda
- Simplify your pub/sub messaging with Amazon SNS message filtering

Contenu connexe

- Features of pub/sub messaging

Réessayez avec le schéma de désactivation

Intention

Le modèle de nouvelle tentative avec interruption améliore la stabilité de l'application en réessayant de manière transparente les opérations qui échouent en raison d'erreurs transitoires.

Motivation

Dans les architectures distribuées, les erreurs transitoires peuvent être causées par une limitation des services, une perte temporaire de connectivité réseau ou une indisponibilité temporaire du service. Réessayer automatiquement les opérations qui échouent en raison de ces erreurs transitoires améliore l'expérience utilisateur et la résilience des applications. Toutefois, des tentatives fréquentes peuvent surcharger la bande passante du réseau et provoquer des conflits. L'interruption exponentielle est une technique dans laquelle les opérations sont relancées en augmentant les temps d'attente pour un nombre spécifié de tentatives.

Applicabilité

Utilisez le schéma de réessai avec retour en arrière dans les cas suivants :

- Vos services limitent fréquemment la demande pour éviter toute surcharge, ce qui entraîne une429 Trop de demandesexception au processus d'appel.
- Le réseau joue un rôle invisible dans les architectures distribuées, et les problèmes temporaires du réseau entraînent des défaillances.
- Le service appelé est temporairement indisponible, ce qui entraîne des défaillances. Les tentatives fréquentes peuvent entraîner une dégradation du service à moins que vous n'introduisiez un délai d'attente en utilisant ce modèle.

Enjeux et considérations

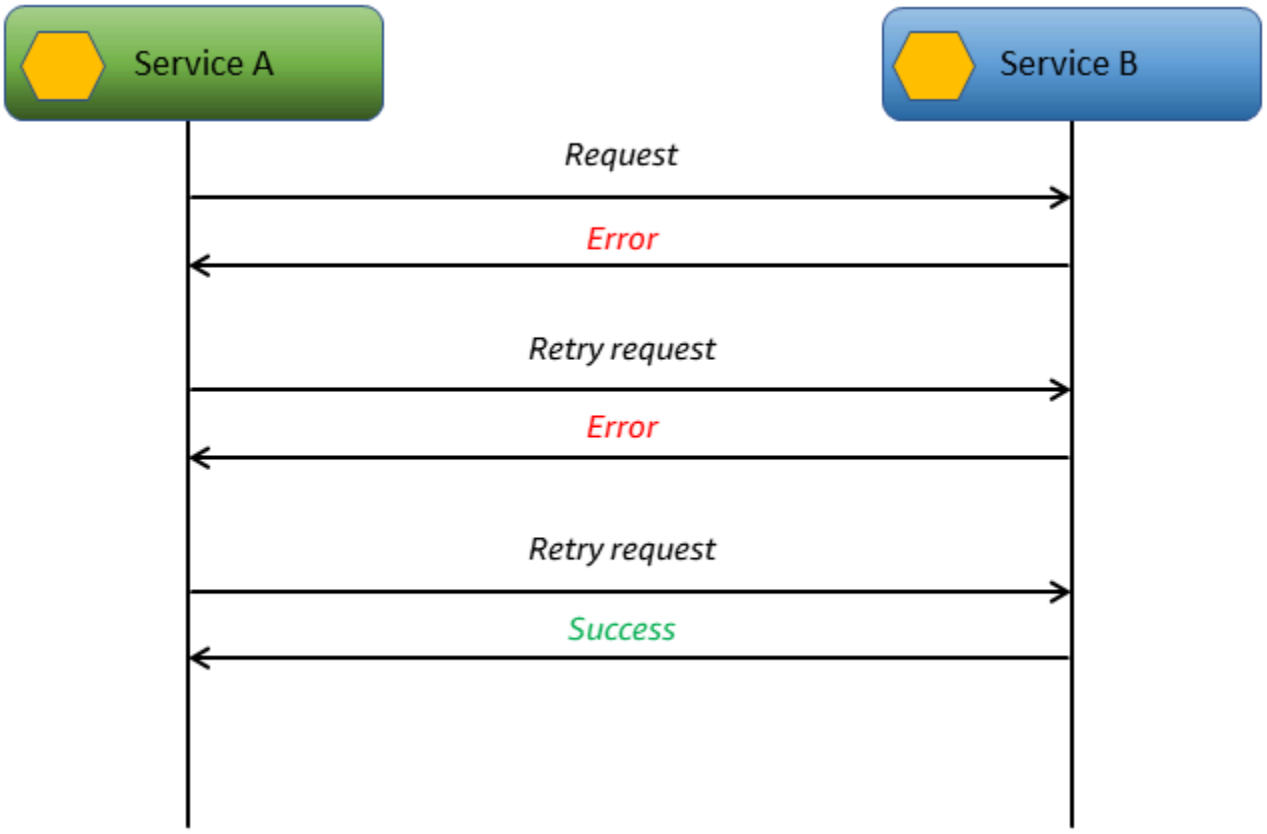
- Idempotence: Si plusieurs appels à la méthode ont le même effet qu'un seul appel sur l'état du système, l'opération est considérée comme idempotente. Les opérations doivent être idempotentes lorsque vous utilisez le schéma de nouvelle tentative avec recul. Dans le cas contraire, des mises à jour partielles risquent d'altérer l'état du système.

- Bande passante réseau: Une dégradation du service peut se produire si trop de nouvelles tentatives occupent la bande passante du réseau, ce qui ralentit les temps de réponse.
- Scénarios d'échec rapide: Pour les erreurs non transitoires, si vous pouvez déterminer la cause de la panne, il est plus efficace de tomber en panne rapidement en utilisant le schéma des disjoncteurs.
- Taux de retrait: L'introduction d'une interruption exponentielle peut avoir un impact sur le délai d'expiration du service, ce qui allonge les délais d'attente pour l'utilisateur final.

Mise en œuvre

Architecture de haut niveau

Le schéma suivant montre comment le service A peut réessayer d'appeler le service B jusqu'à ce qu'une réponse positive soit renvoyée. Si le service B ne renvoie pas de réponse satisfaisante après quelques essais, le service A peut arrêter de réessayer et renvoyer un message d'échec à son appelant.

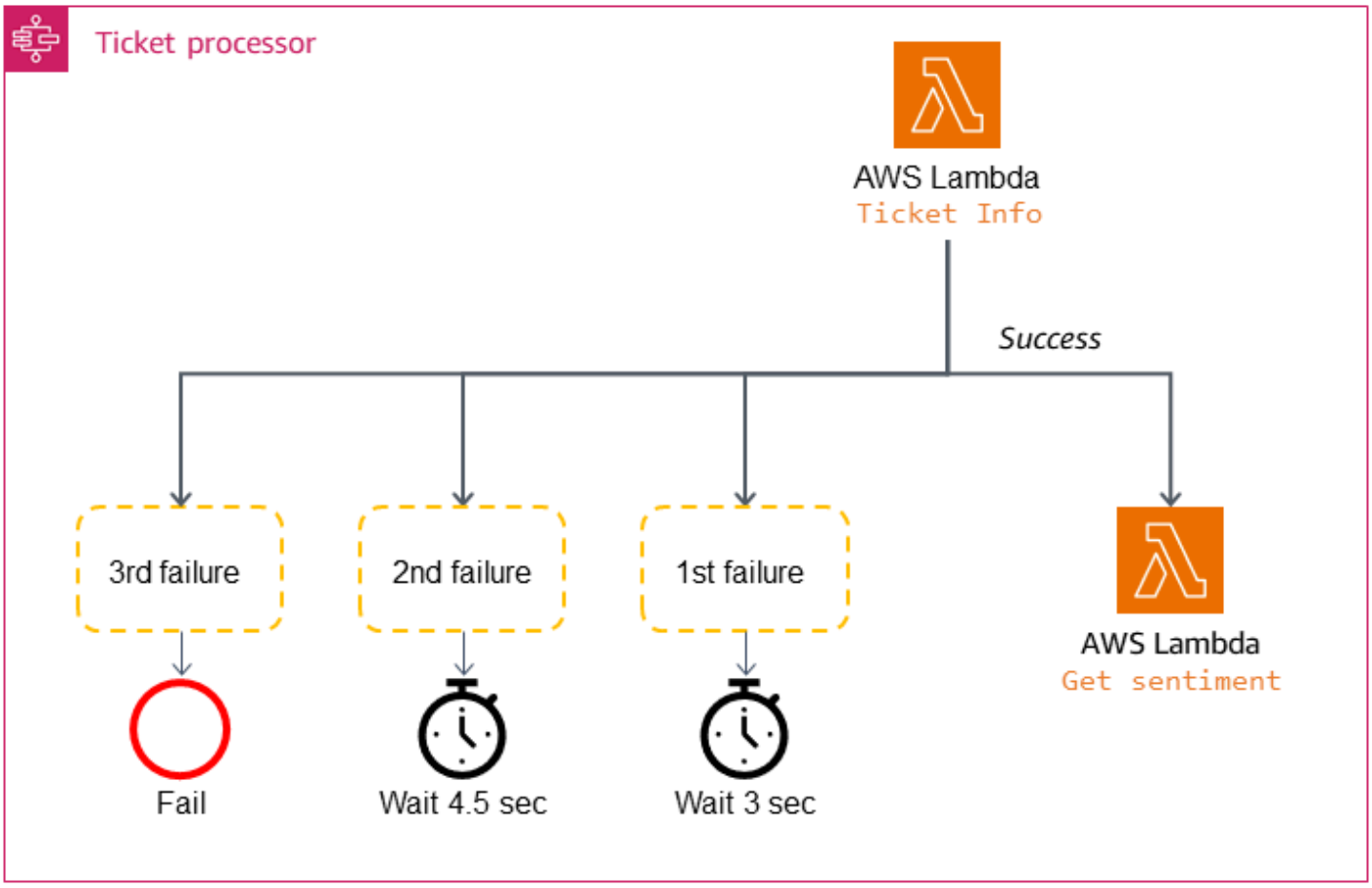


Mise en œuvre en utilisantAWSservices

Le schéma suivant montre un flux de travail de traitement des tickets sur une plateforme de support client. Les tickets provenant de clients mécontents sont accélérés grâce à l'augmentation automatique de la priorité des tickets. Le`Ticket info`La fonction Lambda extrait les détails du ticket et appelle`Get sentiment`Fonction Lambda. Le`Get sentiment`La fonction Lambda vérifie les sentiments des clients en transmettant la description àAmazon Comprehend(non illustré).

Si l'appel au`Get sentiment`La fonction Lambda échoue, le flux de travail réessaie l'opération trois fois.AWS Step Functionspermet un recul exponentiel en vous permettant de configurer la valeur d'intervalle.

Dans cet exemple, un maximum de trois tentatives sont configurées avec un multiplicateur d'augmentation de 1,5 seconde. Si la première tentative a lieu au bout de 3 secondes, la deuxième au bout de 3 x 1,5 seconde = 4,5 secondes et la troisième au bout de 4,5 x 1,5 seconde = 6,75 secondes. Si la troisième tentative échoue, le flux de travail échoue. La logique de sauvegarde ne nécessite aucun code personnalisé. Elle est fournie sous forme de configuration parAWS Step Functions.



Exemple de code

Le code suivant montre l'implémentation du modèle de nouvelle tentative avec backoff.

```
public async Task DoRetriesWithBackOff()
  {
     int retries = 0;
     bool retry;
     do
     {
       //Sample object for sending parameters
       var parameterObj = new InputParameter { SimulateTimeout = "false" };
       var content = new StringContent(JsonConvert.SerializeObject(parameterObj),
                                System.Text.Encoding.UTF8, "application/json");
       var waitInMilliseconds = Convert.ToInt32((Math.Pow(2, retries) - 1) * 100);
       System.Threading.Thread.Sleep(waitInMilliseconds);
       var response =  await _client.PostAsync(_baseURL, content);
       switch (response.StatusCode)
```

```
      {
        //Success
        case HttpStatusCode.OK:
          retry = false;
          Console.WriteLine(response.Content.ReadAsStringAsync().Result);
          break;
        //Throttling, timeouts
        case HttpStatusCode.TooManyRequests:
        case HttpStatusCode.GatewayTimeout:
          retry = true;
          break;
        //Some other error occured, so stop calling the API
        default:
          retry = false;
          break;
      }
      retries++;
    } while (retry && retries < MAX_RETRIES);
  }
```

GitHubréférentiel

Pour une implémentation complète de l'exemple d'architecture pour ce modèle, consultez leGitHubdépôt surhttps://github.com/aws-samples/retry-with-backoff.

Contenu connexe

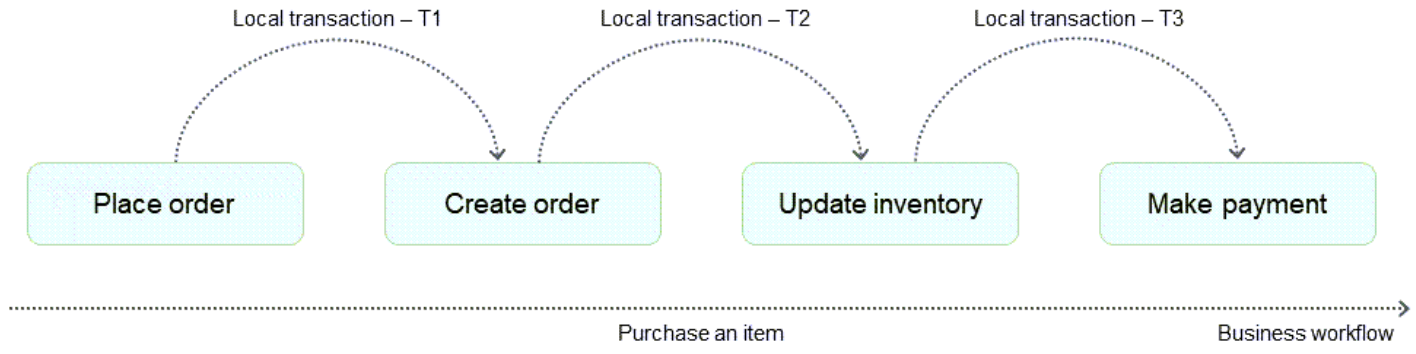
- Délais d'attente, nouvelles tentatives et retour en arrière avec instabilité(Bibliothèque Amazon Builders)

Modèles saga

Une saga consiste en une séquence de transactions locales. Chaque transaction locale d'une saga met à jour la base de données et déclenche la transaction locale suivante. Si une transaction échoue, la saga exécute des transactions de compensation pour annuler les modifications apportées à la base de données par les transactions précédentes.

Cette séquence de transactions locales permet de mettre en place un flux de travail métier en utilisant les principes de continuité et de compensation. Le principe de continuité détermine la reprise du flux de travail vers l'avant, tandis que le principe de compensation détermine la reprise vers l'arrière. Si la mise à jour échoue à n'importe quelle étape de la transaction, la saga publie un événement pour la poursuite (pour réessayer la transaction) ou pour la compensation (pour revenir à l'état des données précédent). Cela garantit le maintien de l'intégrité des données et la cohérence entre les magasins de données.

Par exemple, lorsqu'un utilisateur achète un livre auprès d'un détaillant en ligne, le processus consiste en une séquence de transactions, telles que la création de la commande, la mise à jour du stock, le paiement et l'expédition, qui représente un flux de travail métier. Afin de terminer ce flux de travail, l'architecture distribuée émet une séquence de transactions locales pour créer une commande dans la base de données des commandes, mettre à jour la base de données d'inventaire et mettre à jour la base de données des paiements. Lorsque le processus aboutit, ces transactions sont invoquées de manière séquentielle pour terminer le flux de travail métier, comme le montre le schéma suivant. Toutefois, si l'une de ces transactions locales échoue, le système devrait être en mesure de décider de la prochaine étape appropriée, c'est-à-dire une reprise en avant ou en arrière.



Les deux scénarios suivants permettent de déterminer si l'étape suivante est la reprise en avant ou la reprise en arrière :

- Défaillance au niveau de la plateforme, lorsque quelque chose ne va pas dans l'infrastructure sous-jacente et entraîne l'échec de la transaction. Dans ce cas, le modèle de saga peut effectuer une reprise vers l'avant en réessayant la transaction locale et en poursuivant le processus métier.
- Défaillance au niveau de l'application, lorsque le service de paiement échoue en raison d'un paiement non valide. Dans ce cas, le modèle de saga peut effectuer une reprise vers l'arrière en émettant une transaction compensatoire pour mettre à jour l'inventaire et les bases de données de commandes, et rétablir leur état antérieur.

Le modèle de saga gère le flux de travail métier et garantit que l'état final souhaité est atteint grâce à une reprise vers l'avant. En cas d'échec, il annule les transactions locales en utilisant la reprise vers l'arrière pour éviter les problèmes de cohérence des données.

Le modèle de saga comporte deux variantes : chorégraphie et orchestration.

Chorégraphie de saga

Le modèle de chorégraphie de saga dépend des événements diffusés par les microservices. Les participants à la saga (microservices) s'abonnent aux événements et agissent en fonction des déclencheurs des événements. Par exemple, le service de commande décrit dans le schéma suivant émet un événement `OrderPlaced`. Le service d'inventaire s'abonne à cet événement et met à jour l'inventaire lorsque l'événement `OrderPlaced` est émis. De même, les services des participants agissent en fonction du contexte de l'événement émis.

Le modèle de chorégraphie de saga convient lorsque la saga ne compte que quelques participants et que vous avez besoin d'une implémentation simple, sans point de défaillance unique. Lorsque d'autres participants sont ajoutés, il devient plus difficile de suivre les dépendances entre les participants en utilisant ce modèle.

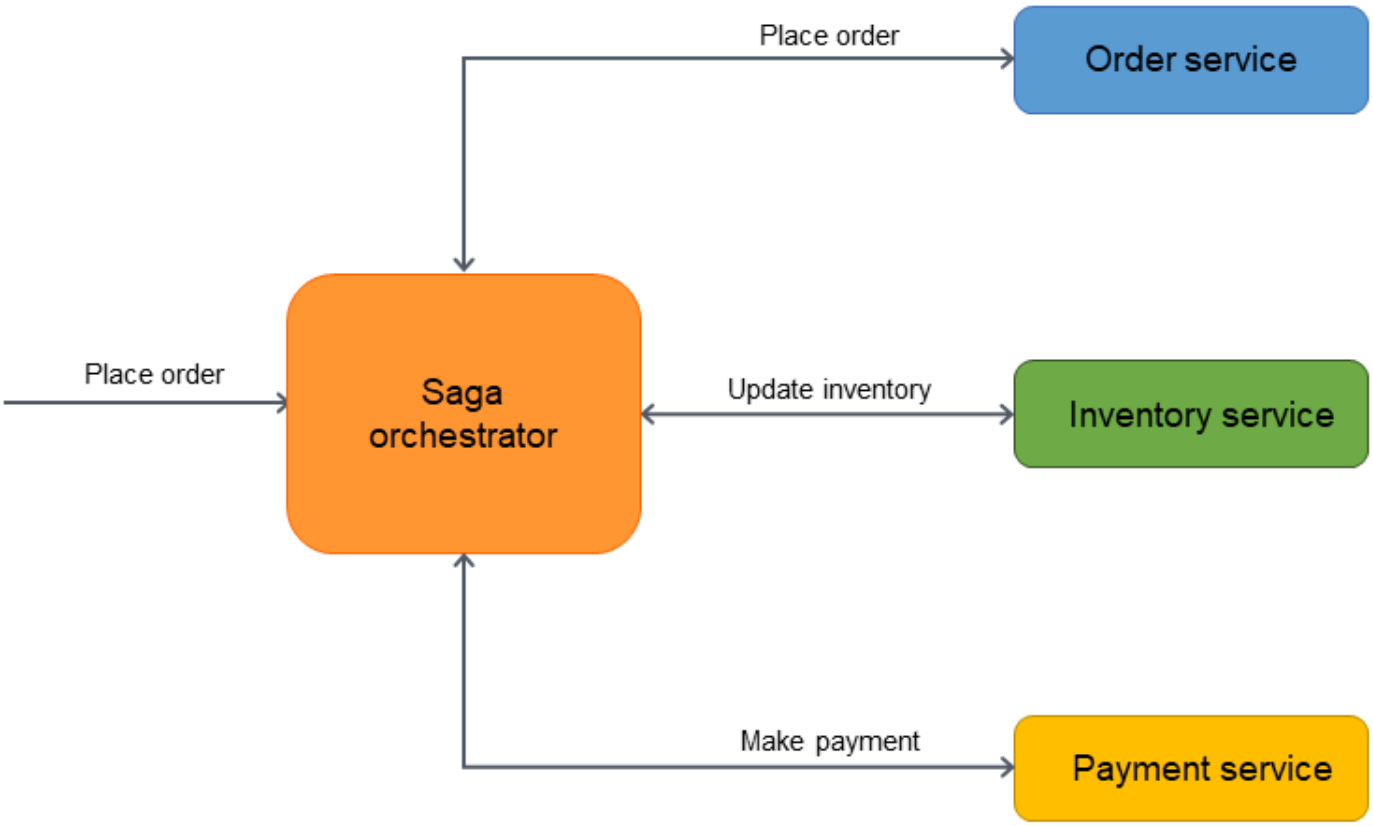
Pour un examen détaillé, veuillez consulter la section Saga choreography de ce guide.

Orchestration de saga

Le modèle d'orchestration de saga a un coordinateur central appelé orchestrateur. L'orchestrateur de saga gère et coordonne l'ensemble du cycle de vie des transactions. Il est conscient de la série

d'étapes à suivre pour terminer la transaction. Pour exécuter une étape, il envoie un message au microservice du participant pour qu'il effectue l'opération. Le microservice du participant termine l'opération et renvoie un message à l'orchestrateur. Sur la base du message qu'il reçoit, l'orchestrateur décide quel microservice exécuter ensuite dans la transaction.

Le modèle d'orchestration de saga convient lorsqu'il y a de nombreux participants, et qu'un couplage faible est nécessaire entre les participants à la saga. L'orchestrateur résume la complexité de la logique en associant les participants de manière faible. Cependant, l'orchestrateur peut devenir un point de défaillance unique, car il contrôle l'ensemble du flux de travail.



Pour un examen détaillé, veuillez consulter la section Saga orchestration de ce guide.

Modèle de chorégraphie de saga

Intention

Le modèle de chorégraphie de saga permet de préserver l'intégrité des données dans les transactions distribuées qui couvrent plusieurs services en utilisant des abonnements à des événements. Dans une transaction distribuée, plusieurs services peuvent être appelés avant qu'une

transaction ne soit terminée. Lorsque les services stockent des données dans différents magasins de données, il peut être difficile de maintenir la cohérence des données entre ces magasins de données.

Motivation

Une transaction est une unité de travail unique qui peut impliquer plusieurs étapes, toutes les étapes étant entièrement exécutées ou aucune étape n'étant exécutée, ce qui permet à un magasin de données de conserver son état cohérent. Les termes atomicité, cohérence, isolement et durabilité (ACID) définissent les propriétés d'une transaction. Les bases de données relationnelles fournissent des transactions ACID pour maintenir la cohérence des données.

Pour maintenir la cohérence d'une transaction, les bases de données relationnelles utilisent la méthode de validation en deux phases (2PC). Il s'agit d'une phase de préparation et d'une phase de validation.

- Au cours de la phase de préparation, le processus de coordination demande aux processus participants à la transaction (participants) de promettre de valider ou d'annuler la transaction.
- Dans la phase de validation, le processus de coordination demande aux participants de valider la transaction. Si les participants ne parviennent pas à s'engager lors de la phase de préparation, la transaction est annulée.

Dans les systèmes distribués qui suivent un modèle de database-per-service conception, la validation en deux phases n'est pas une option. En effet, chaque transaction est distribuée entre différentes bases de données et aucun contrôleur ne peut coordonner un processus similaire à la validation en deux phases dans les magasins de données relationnels. Dans ce cas, une solution consiste à utiliser le modèle de chorégraphie de saga.

Applicabilité

Utilisez le modèle de chorégraphie de saga lorsque :

- Votre système a besoin de l'intégrité et de la cohérence des données dans les transactions distribuées qui s'étendent sur plusieurs magasins de données.
- Le magasin de données (par exemple, une base de données NoSQL) ne propose pas le 2PC pour fournir des transactions ACID, vous devez mettre à jour plusieurs tables au cours d'une seule transaction, et l'implémentation du 2PC dans les limites de l'application serait une tâche complexe.
- Un processus de contrôle central qui gère les transactions des participants pourrait devenir un point de défaillance unique.

- Les participants à la saga sont des services indépendants et doivent être couplés de manière faible.
- Il existe une communication entre des contextes délimités dans un domaine métier.

Problèmes et considérations

- Complexité : à mesure que le nombre de microservices augmente, la chorégraphie de saga peut devenir difficile à gérer en raison du nombre d'interactions entre les microservices. En outre, les transactions compensatoires et les nouvelles tentatives ajoutent de la complexité au code de l'application, ce qui peut entraîner des frais de maintenance. La chorégraphie convient lorsque la saga ne compte que quelques participants et que vous avez besoin d'une implémentation simple, sans point de défaillance unique. Lorsque d'autres participants sont ajoutés, il devient plus difficile de suivre les dépendances entre les participants en utilisant ce modèle.
- Mise en œuvre résiliente : dans la chorégraphie de saga, il est plus difficile d'implémenter des délais d'attente, des nouvelles tentatives et d'autres modèles de résilience à l'échelle mondiale que dans le cas d'une orchestration de saga. La chorégraphie doit être mise en œuvre sur des composants individuels plutôt qu'au niveau de l'orchestrateur.
- Dépendances cycliques : les participants consomment des messages diffusés les uns par les autres. Cela peut entraîner des dépendances cycliques, puis des complexités du code et des frais de maintenance, ainsi que d'éventuels blocages.
- Problème d'écritures doubles : le microservice doit mettre à jour la base de données de manière atomique et diffuser un événement. L'échec de l'une ou l'autre opération peut entraîner un état incohérent. Une façon de résoudre ce problème consiste à utiliser le modèle de boîte d'envoi transactionnelle.
- Préservation des événements : les participants à la saga agissent en fonction des événements diffusés. Il est important d'enregistrer les événements dans l'ordre dans lequel ils se produisent à des fins d'audit, de débogage et de rediffusion. Vous pouvez utiliser le modèle d'approvisionnement d'événement pour conserver les événements dans un magasin d'événements au cas où une rediffusion de l'état du système serait nécessaire pour rétablir la cohérence des données. Les magasins d'événements peuvent également être utilisés à des fins d'audit et de dépannage, car ils reflètent chaque modification apportée au système.
- Cohérence à terme : le traitement séquentiel des transactions locales aboutit à une cohérence à terme, ce qui peut être un défi dans les systèmes nécessitant une forte cohérence. Vous pouvez résoudre ce problème en définissant les attentes de vos équipes métier en ce qui concerne le

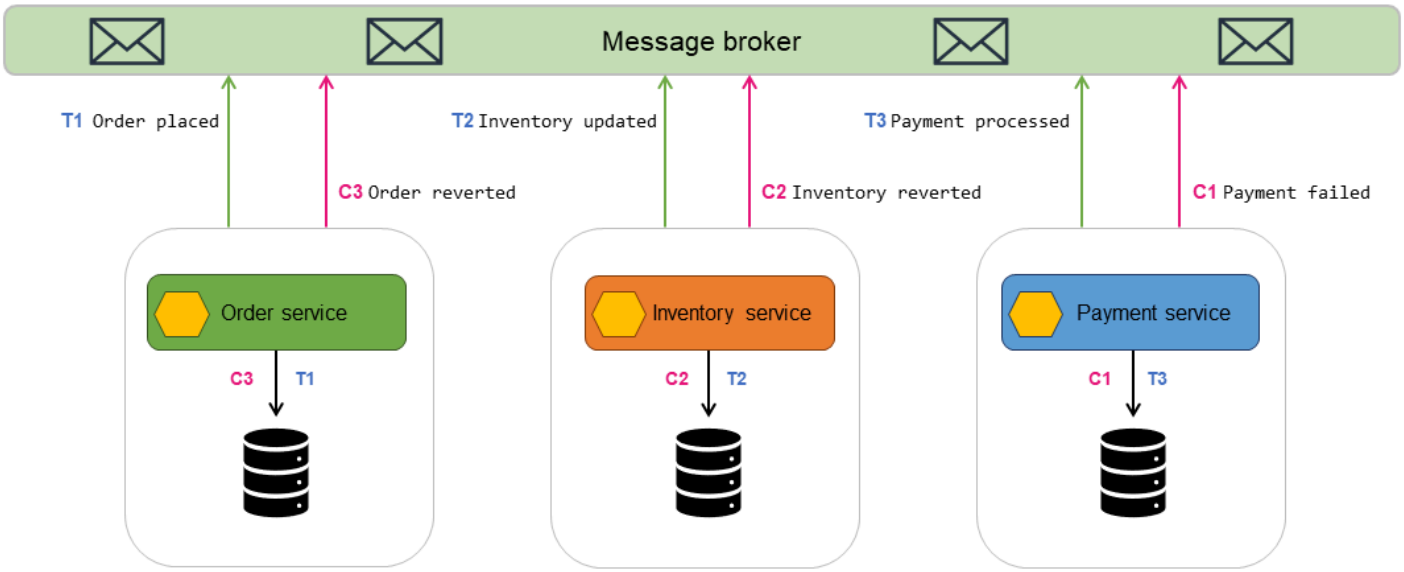
modèle de cohérence ou en réévaluant le cas d'utilisation et en optant pour une base de données offrant une forte cohérence.

- Idempotence : les participants à la saga doivent être idempotents pour permettre des exécutions répétées en cas de défaillances transitoires causées par des pannes inattendues ou des défaillances de l'orchestrateur.
- Isolement des transactions : le modèle de saga ne permet pas d'isoler les transactions, ce qui est l'une des quatre propriétés des transactions ACID. Le degré d'isolement d'une transaction détermine dans quelle mesure les autres transactions simultanées peuvent affecter les données sur lesquelles la transaction opère. L'orchestration simultanée de transactions peut entraîner l'obsolescence des données. Nous vous recommandons d'utiliser le verrouillage sémantique pour gérer de tels scénarios.
- Observabilité : l'observabilité fait référence à la journalisation et au suivi détaillés pour résoudre les problèmes liés au processus de mise en œuvre et d'orchestration. Cela devient important lorsque le nombre de participants à la saga augmente, ce qui complique le débogage. nd-to-end La surveillance électronique et le reporting sont plus difficiles à réaliser dans le cadre d'une chorégraphie de saga que dans le cas d'une orchestration de saga.
- Problèmes de latence : les transactions compensatoires peuvent ajouter de la latence au temps de réponse global lorsque la saga comporte plusieurs étapes. Si les transactions émettent des appels synchrones, cela peut encore augmenter la latence.

Mise en œuvre

Architecture de haut niveau

Dans le schéma d'architecture suivant, la chorégraphie de saga compte trois participants : le service de commande, le service d'inventaire et le service de paiement. Trois étapes sont nécessaires pour terminer la transaction : T1, T2 et T3. Trois transactions compensatoires rétablissent les données dans leur état initial : C1, C2 et C3.



- Le service de commande exécute une transaction locale, T1, qui met à jour de manière atomique la base de données et diffuse un message `Order placed` à l'agent de messages.
- Le service d'inventaire s'abonne aux messages du service de commande et reçoit le message indiquant qu'une commande a été créée.
- Le service d'inventaire exécute une transaction locale, T2, qui met à jour de manière atomique la base de données et diffuse un message `Inventory updated` à l'agent de messages.
- Le service de paiement s'abonne aux messages du service d'inventaire et reçoit le message indiquant que l'inventaire a été mis à jour.
- Le service de paiement exécute une transaction locale, T3, qui met à jour de manière atomique la base de données avec les informations de paiement et diffuse un message `Payment processed` à l'agent de messages.
- Si le paiement échoue, le service de paiement exécute une transaction compensatoire, C1, qui annule le paiement de manière atomique dans la base de données et diffuse un message `Payment failed` à l'agent de messages.
- Les transactions compensatoires C2 et C3 sont exécutées pour rétablir la cohérence des données.

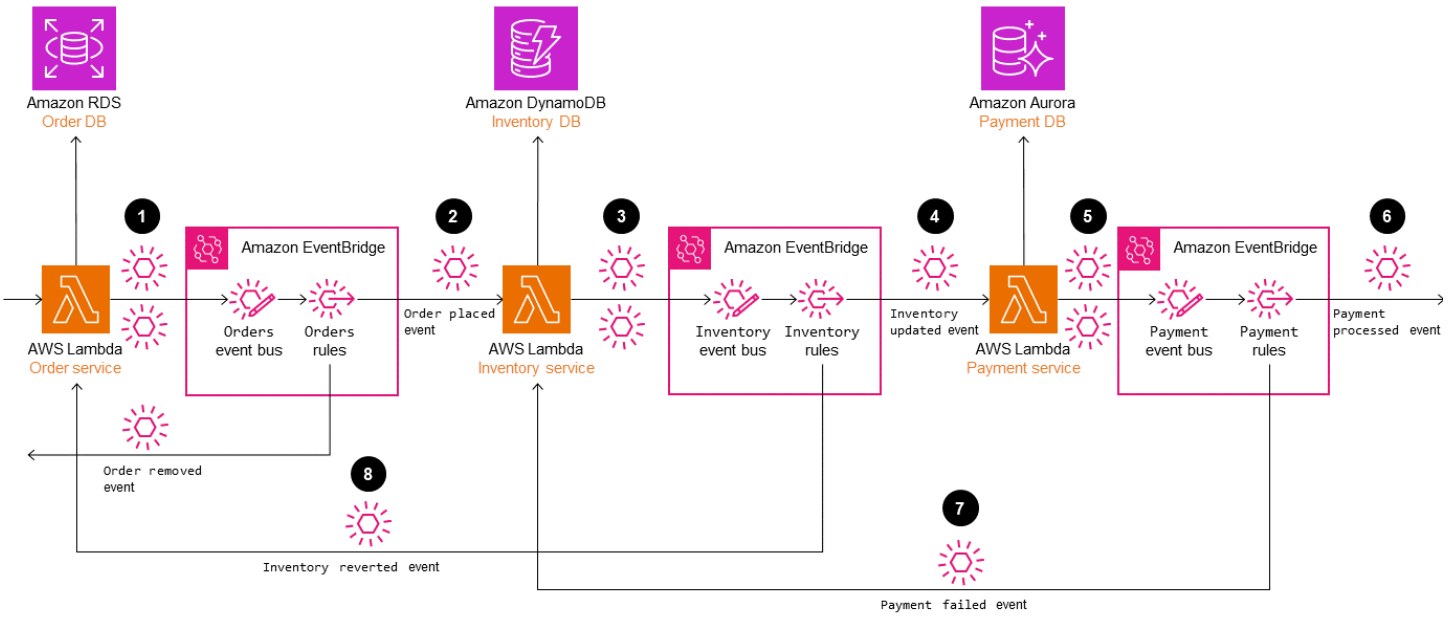
Mise en œuvre à l'aide des services AWS

Vous pouvez implémenter le modèle de chorégraphie de la saga en utilisant Amazon EventBridge. EventBridgeutilise des événements pour connecter les composants de l'application. Il traite les événements par le biais de bus ou de pipelines d'événements. Un bus d'événements est un routeur

qui reçoit des événements et les transmet à zéro ou plusieurs destinations, ou cibles. Les règles associées au bus d'événements évaluent les événements au fur et à mesure qu'ils arrivent et les envoient aux cibles pour traitement.

Dans l'architecture suivante :

- Les microservices (service de commande, service d'inventaire et service de paiement) sont implémentés sous forme de fonctions Lambda.
- Il existe trois EventBridge bus personnalisés : le bus `Orders` d'événement, le bus d'`Inventory`événement et le bus `Payment` d'événement.
- Les règles `Orders`, `Inventory` et `Payment` correspondent aux événements envoyés au bus d'événements correspondant et invoquent les fonctions Lambda.



Dans un scénario réussi, lorsqu'une commande est passée :

1. Le service de commande traite la demande et envoie l'événement au bus d'événements `Orders`.
2. Les règles `Orders` correspondent aux événements et démarrent le service d'inventaire.
3. Le service d'inventaire met à jour l'inventaire et envoie l'événement au bus d'événements `Inventory`.
4. Les règles `Inventory` correspondent aux événements et démarrent le service de paiement.

5. Le service de paiement traite le paiement et envoie l'événement au bus d'événements `Payment`.
6. Les règles `Payment` correspondent aux événements et envoient la notification de l'événement `Payment processed` à l'écouteur.

   Par ailleurs, en cas de problème dans le traitement des commandes, les EventBridge règles lancent les transactions compensatoires pour annuler les mises à jour des données afin de maintenir la cohérence et l'intégrité des données.
7. Si le paiement échoue, les règles `Payment` traitent l'événement et démarrent le service d'inventaire.Le service d'inventaire exécute des transactions compensatoires pour rétablir l'inventaire.
8. Lorsque l'inventaire a été rétabli, le service d'inventaire envoie l'événement `Inventory reverted` au bus d'événements `Inventory`.Cet événement est traité par des règles `Inventory`.Il lance le service de commande, qui exécute la transaction compensatoire pour supprimer la commande.

Contenu connexe

- Saga orchestration pattern
- Transactional outbox pattern
- Retry with backoff pattern

Modèle d'orchestration de saga

Intention

Le modèle d'orchestration de saga utilise un coordinateur central (orchestrateur) pour aider à préserver l'intégrité des données dans les transactions distribuées qui couvrent plusieurs services. Dans une transaction distribuée, plusieurs services peuvent être appelés avant qu'une transaction ne soit terminée. Lorsque les services stockent des données dans différents magasins de données, il peut être difficile de maintenir la cohérence des données entre ces magasins de données.

Motivation

Une transaction est une unité de travail unique qui peut impliquer plusieurs étapes, toutes les étapes étant entièrement exécutées ou aucune étape n'étant exécutée, ce qui permet à un magasin de

données de conserver son état cohérent. Les termes atomicité, cohérence, isolement et durabilité (ACID) définissent les propriétés d'une transaction. Les bases de données relationnelles fournissent des transactions ACID pour maintenir la cohérence des données.

Pour maintenir la cohérence d'une transaction, les bases de données relationnelles utilisent la méthode de validation en deux phases (2PC). Il s'agit d'une phase de préparation et d'une phase de validation.

- Au cours de la phase de préparation, le processus de coordination demande aux processus participants à la transaction (participants) de promettre de valider ou d'annuler la transaction.
- Dans la phase de validation, le processus de coordination demande aux participants de valider la transaction. Si les participants ne parviennent pas à s'engager lors de la phase de préparation, la transaction est annulée.

Dans les systèmes distribués qui suivent un modèle de database-per-service conception, la validation en deux phases n'est pas une option. En effet, chaque transaction est distribuée entre différentes bases de données et aucun contrôleur ne peut coordonner un processus similaire à la validation en deux phases dans les magasins de données relationnels. Dans ce cas, une solution consiste à utiliser le modèle d'orchestration de saga.

Applicabilité

Utilisez le modèle d'orchestration de saga lorsque :

- Votre système a besoin de l'intégrité et de la cohérence des données dans les transactions distribuées qui s'étendent sur plusieurs magasins de données.
- Le magasin de données ne propose pas de 2PC pour les transactions ACID, et que la mise en œuvre de 2PC dans les limites de l'application est une tâche complexe.
- Vous avez des bases de données NoSQL qui ne fournissent pas de transactions ACID et vous devez mettre à jour plusieurs tables au cours d'une seule transaction.

Problèmes et considérations

- Complexité : les transactions compensatoires et les nouvelles tentatives ajoutent de la complexité au code de l'application, ce qui peut entraîner des frais de maintenance.
- Cohérence à terme : le traitement séquentiel des transactions locales aboutit à une cohérence à terme, ce qui peut être un défi dans les systèmes nécessitant une forte cohérence. Vous pouvez

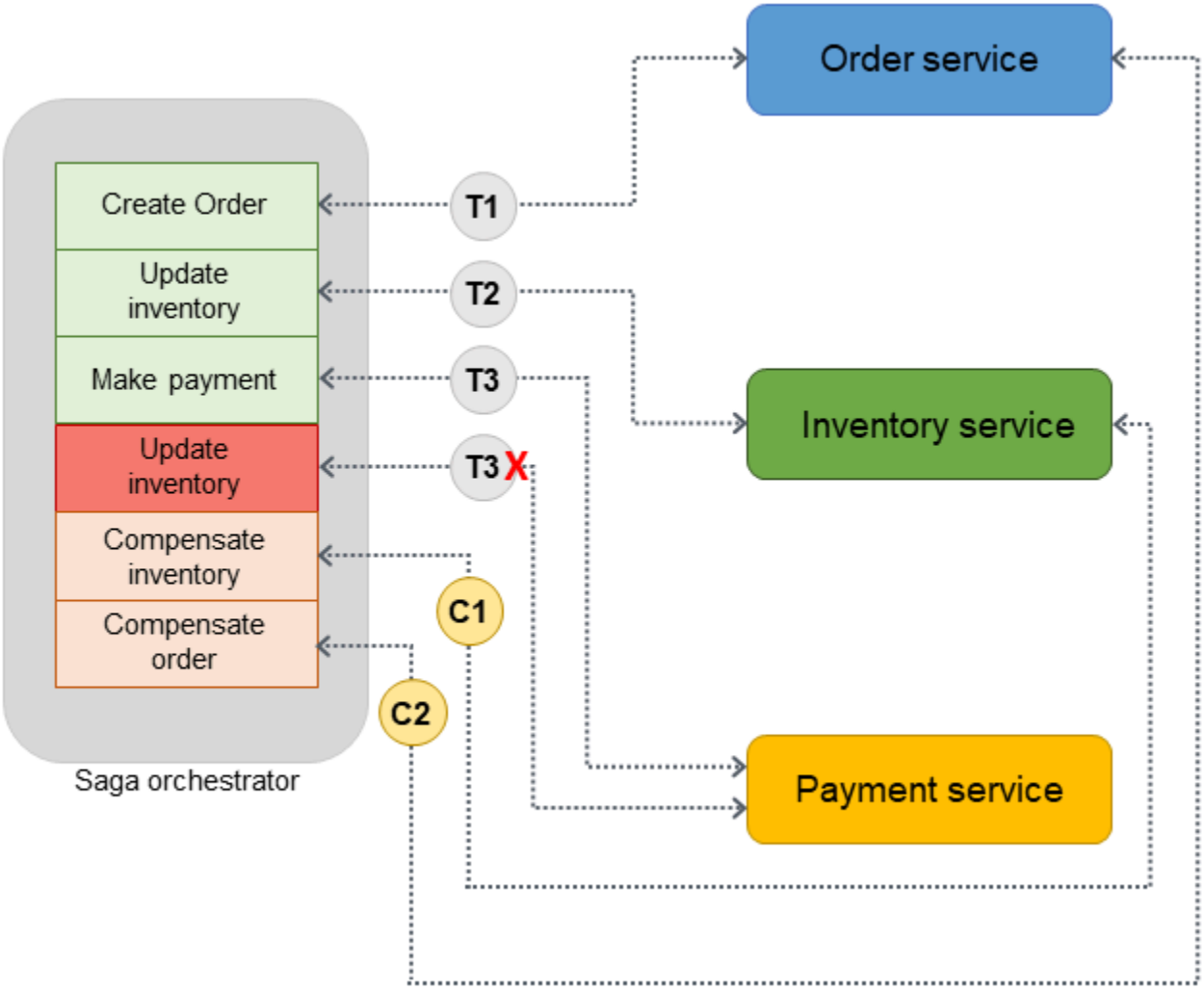
résoudre ce problème en définissant les attentes de vos équipes métier en ce qui concerne le modèle de cohérence ou en optant pour un magasin de données offrant une forte cohérence.

- Idempotence : les participants à la saga doivent être idempotents pour permettre des exécutions répétées en cas de défaillances transitoires causées par des pannes inattendues ou des défaillances de l'orchestrateur.
- Isolement des transactions : la saga n'est pas isolée des transactions. L'orchestration simultanée de transactions peut entraîner l'obsolescence des données. Nous vous recommandons d'utiliser le verrouillage sémantique pour gérer de tels scénarios.
- Observabilité : l'observabilité fait référence à la journalisation et au suivi détaillés pour résoudre les problèmes liés au processus d'exécution et d'orchestration. Cela devient important lorsque le nombre de participants à la saga augmente, ce qui complique le débogage.
- Problèmes de latence : les transactions compensatoires peuvent ajouter de la latence au temps de réponse global lorsque la saga comporte plusieurs étapes. Évitez les appels synchrones dans de tels cas.
- Point de défaillance unique : l'orchestrateur peut devenir un point de défaillance unique, car il coordonne l'ensemble de la transaction. Dans certains cas, le modèle de chorégraphie de saga est préféré en raison de ce problème.

Mise en œuvre

Architecture de haut niveau

Dans le schéma d'architecture suivant, l'orchestrateur de saga compte trois participants : le service de commande, le service d'inventaire et le service de paiement. Trois étapes sont nécessaires pour terminer la transaction : T1, T2 et T3. L'orchestrateur de saga connaît les étapes et les exécute dans l'ordre requis. Lorsque l'étape T3 échoue (échec de paiement), l'orchestrateur exécute les transactions compensatoires C1 et C2 pour rétablir les données dans leur état initial.



Vous pouvez utiliser AWS Step Functions pour implémenter l'orchestration de saga lorsque la transaction est distribuée sur plusieurs bases de données.

Mise en œuvre à l'aide des services AWS

L'exemple de solution utilise le flux de travail standard de Step Functions pour implémenter le modèle d'orchestration de saga.

Lorsqu'un client appelle l'API, la fonction Lambda est invoquée et le prétraitement est effectué dans la fonction Lambda. La fonction lance le flux de travail Step Functions pour commencer à traiter la transaction distribuée. Si aucun prétraitement n'est requis, vous pouvez lancer le flux de travail Step Functions directement depuis API Gateway sans utiliser la fonction Lambda.

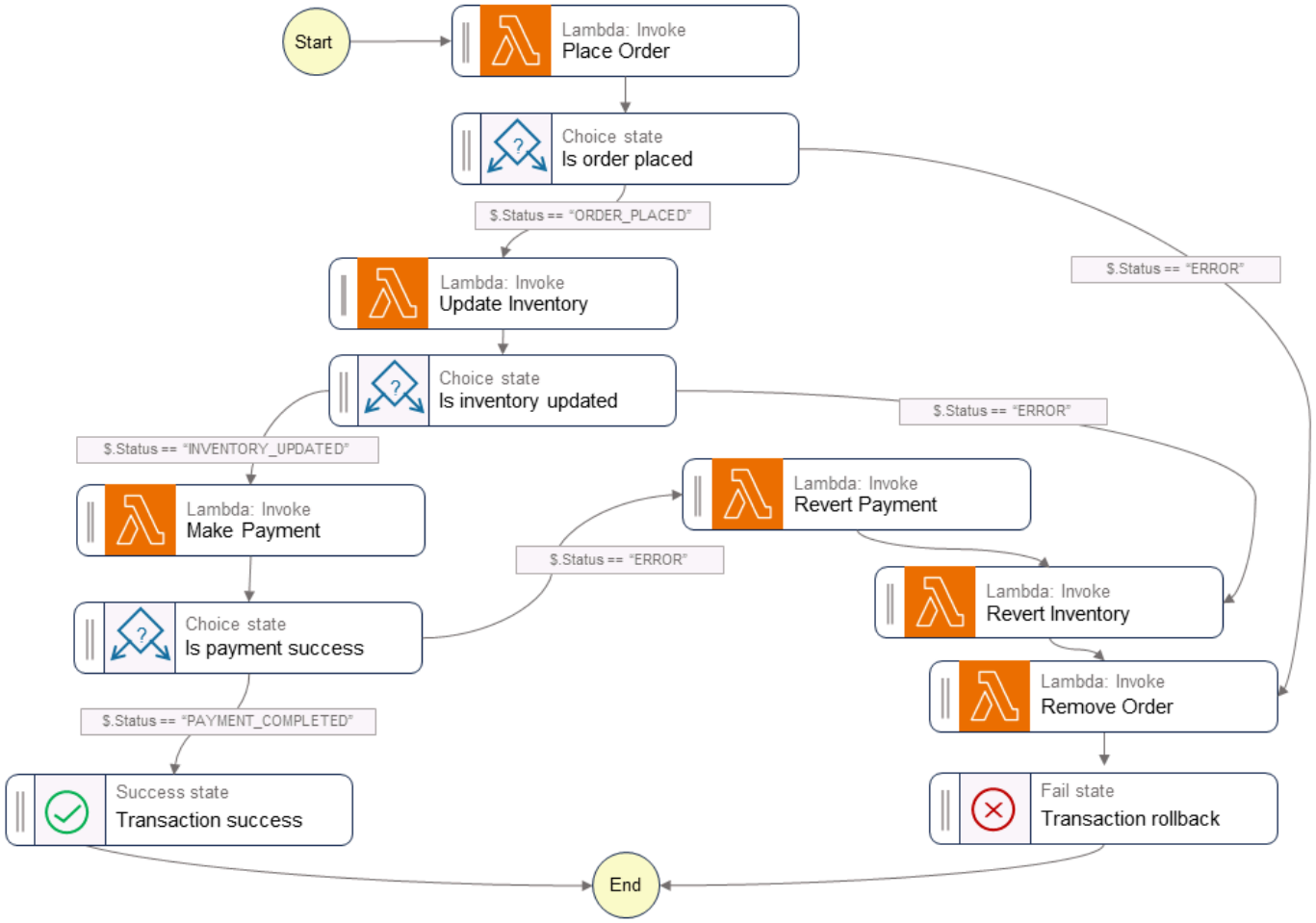
L'utilisation de Step Functions atténue le problème du point de défaillance unique, inhérent à la mise en œuvre du modèle d'orchestration de saga. Step Functions intègre une tolérance aux pannes et maintient la capacité de service dans plusieurs zones de disponibilité dans chaque Région AWS afin de protéger les applications contre les pannes individuelles de machines ou de centres de données. Cela permet de garantir une haute disponibilité à la fois pour le service lui-même et pour le flux de travail des applications qu'il gère.

Flux de travail Step Functions

La machine d'état Step Functions vous permet de configurer les exigences de flux de contrôle basées sur les décisions pour la mise en œuvre du modèle. Le flux de travail Step Functions appelle les différents services pour le placement des commandes, la mise à jour de l'inventaire et le traitement des paiements afin de terminer la transaction et envoie une notification d'événement pour un traitement ultérieur. Le flux de travail Step Functions agit en tant qu'orchestrateur pour coordonner les transactions. Si le flux de travail contient des erreurs, l'orchestrateur exécute les transactions compensatoires pour garantir le maintien de l'intégrité des données dans tous les services.

Le schéma suivant illustre les étapes du flux de travail Step Functions. Les étapes `Place Order`, `Update Inventory` et `Make Payment` indiquent le chemin à suivre pour réussir. La commande est passée, l'inventaire est mis à jour et le paiement est traité avant qu'un état `Success` ne soit renvoyé à l'appelant.

Les fonctions Lambda `Revert Payment`, `Revert Inventory` et `Remove Order` indiquent les transactions compensatoires que l'orchestrateur exécute en cas d'échec d'une étape du flux de travail. Si le flux de travail échoue à l'étape `Update Inventory`, l'orchestrateur appelle les étapes `Revert Inventory` et `Remove Order` avant de renvoyer un état `Fail` à l'appelant. Ces transactions compensatoires garantissent le maintien de l'intégrité des données. L'inventaire revient à son niveau initial et la commande est annulée.



Exemple de code

L'exemple de code suivant montre comment créer un orchestrateur de saga à l'aide de Step Functions. Pour consulter le code complet, consultez le GitHubréférentiel de cet exemple.

Définitions de tâche

```
var successState = new Succeed(this,"SuccessState");
var failState = new Fail(this, "Fail");

var placeOrderTask = new LambdaInvoke(this, "Place Order", new LambdaInvokeProps
{
    LambdaFunction = placeOrderLambda,
    Comment = "Place Order",
    RetryOnServiceExceptions = false,
    PayloadResponseOnly = true
});
```

```
var updateInventoryTask = new LambdaInvoke(this,"Update Inventory", new
 LambdaInvokeProps
{
    LambdaFunction = updateInventoryLambda,
    Comment = "Update inventory",
    RetryOnServiceExceptions = false,
    PayloadResponseOnly = true
});

var makePaymentTask = new LambdaInvoke(this,"Make Payment", new LambdaInvokeProps
{
    LambdaFunction = makePaymentLambda,
    Comment = "Make Payment",
    RetryOnServiceExceptions = false,
    PayloadResponseOnly = true
});

var removeOrderTask = new LambdaInvoke(this, "Remove Order", new LambdaInvokeProps
{
    LambdaFunction = removeOrderLambda,
    Comment = "Remove Order",
    RetryOnServiceExceptions = false,
    PayloadResponseOnly = true
}).Next(failState);

var revertInventoryTask = new LambdaInvoke(this,"Revert Inventory", new
 LambdaInvokeProps
{
    LambdaFunction = revertInventoryLambda,
    Comment = "Revert inventory",
    RetryOnServiceExceptions = false,
    PayloadResponseOnly = true
}).Next(removeOrderTask);

var revertPaymentTask = new LambdaInvoke(this,"Revert Payment", new LambdaInvokeProps
{
    LambdaFunction = revertPaymentLambda,
    Comment = "Revert Payment",
    RetryOnServiceExceptions = false,
    PayloadResponseOnly = true
}).Next(revertInventoryTask);

var waitState = new Wait(this, "Wait state", new WaitProps
```

```
{
    Time = WaitTime.Duration(Duration.Seconds(30))
}).Next(revertInventoryTask);
```

Définitions de Step Functions et des machines d'état

```
var stepDefinition = placeOrderTask
                .Next(new Choice(this, "Is order placed")
                    .When(Condition.StringEquals("$.Status", "ORDER_PLACED"),
 updateInventoryTask
                        .Next(new Choice(this, "Is inventory updated")
                            .When(Condition.StringEquals("$.Status",
 "INVENTORY_UPDATED"),
                                makePaymentTask.Next(new Choice(this, "Is payment
 success")
                                    .When(Condition.StringEquals("$.Status",
 "PAYMENT_COMPLETED"), successState)
                                    .When(Condition.StringEquals("$.Status", "ERROR"),
 revertPaymentTask)))
                            .When(Condition.StringEquals("$.Status", "ERROR"),
 waitState)))
                    .When(Condition.StringEquals("$.Status", "ERROR"), failState));

var stateMachine = new StateMachine(this, "DistributedTransactionOrchestrator", new
 StateMachineProps {
    StateMachineName = "DistributedTransactionOrchestrator",
    StateMachineType = StateMachineType.STANDARD,
    Role = iamStepFunctionRole,
    TracingEnabled = true,
    Definition = stepDefinition
});
```

GitHub référentiel

Pour une implémentation complète de l'exemple d'architecture pour ce modèle, consultez le GitHub référentiel à l'adresse https://github.com/aws-samples/ saga-orchestration-netcore-blog.

Références du blog

- Building a serverless distributed application using Saga Orchestration pattern

Contenu connexe

- Saga choreography pattern
- Transactional outbox pattern

Vidéos

La vidéo suivante explique comment implémenter le modèle d'orchestration de la saga en utilisant AWS Step Functions.

Motif Scatter-Gather

Intention

Le modèle de diffusion est un modèle de routage de messages qui consiste à diffuser des demandes similaires ou connexes à plusieurs destinataires et à agréger leurs réponses en un seul message à l'aide d'un composant appelé agrégateur. Ce modèle permet de réaliser la parallélisation, de réduire la latence de traitement et de gérer les communications asynchrones. Il est simple d'implémenter le modèle de collecte par dispersion en utilisant une approche synchrone, mais une approche plus puissante consiste à l'implémenter sous forme de routage de messages dans une communication asynchrone, avec ou sans service de messagerie.

Motivation

Lors du traitement d'une demande, une demande dont le traitement séquentiel peut prendre du temps peut être divisée en plusieurs demandes traitées en parallèle. Vous pouvez également envoyer des demandes à plusieurs systèmes externes via des appels d'API pour obtenir une réponse. Le modèle de collecte par dispersion est utile lorsque vous avez besoin d'informations provenant de plusieurs sources. Scatter-gather agrège les résultats pour vous aider à prendre une décision éclairée ou à sélectionner la meilleure réponse à la demande.

Le schéma de diffusion comprend deux phases, comme son nom l'indique :

- La phase de diffusion traite le message de demande et l'envoie à plusieurs destinataires en parallèle. Au cours de cette phase, l'application répartit les demandes sur le réseau et continue de s'exécuter sans attendre de réponses immédiates.
- Au cours de la phase de collecte, l'application collecte les réponses des destinataires et les filtre ou les combine en une réponse unifiée. Lorsque toutes les réponses ont été collectées, elles peuvent soit être agrégées en une seule réponse, soit choisir la meilleure réponse pour un traitement ultérieur.

Applicabilité

Utilisez le modèle scatter-gather lorsque :

- Vous prévoyez d'agréger et de consolider les données provenant de différentes API afin de créer une réponse précise. Le modèle consolide les informations provenant de sources disparates en un tout cohérent. Par exemple, un système de réservation peut demander à plusieurs destinataires d'obtenir des devis auprès de plusieurs partenaires externes.
- La même demande doit être envoyée simultanément à plusieurs destinataires pour terminer une transaction. Par exemple, vous pouvez utiliser ce modèle pour interroger les données d'inventaire en parallèle afin de vérifier la disponibilité d'un produit.
- Vous souhaitez mettre en œuvre un système fiable et évolutif dans lequel l'équilibrage de charge peut être réalisé en répartissant les demandes entre plusieurs destinataires. Si un destinataire échoue ou est confronté à une charge de travail élevée, les autres destinataires peuvent toujours traiter les demandes.
- Vous souhaitez optimiser les performances lorsque vous implémentez des requêtes complexes impliquant plusieurs sources de données. Vous pouvez répartir la requête dans les bases de données pertinentes, rassembler les résultats partiels et les combiner pour obtenir une réponse complète.
- Vous implémentez un type de traitement map-reduce dans lequel la demande de données est acheminée vers plusieurs points de terminaison de traitement des données pour le sharding et la réplication. Les résultats partiels sont filtrés et combinés pour composer la bonne réponse.
- Vous souhaitez répartir les opérations d'écriture sur un espace clé de partition dans le cadre de charges de travail intensives en écriture dans les bases de données clé-valeur. L'agrégateur lit les résultats en interrogeant les données de chaque partition, puis les consolide en une seule réponse.

Problèmes et considérations

- Tolérance aux pannes : ce modèle repose sur plusieurs destinataires qui travaillent en parallèle. Il est donc essentiel de gérer les défaillances avec élégance. Pour atténuer l'impact des défaillances des destinataires sur l'ensemble du système, vous pouvez mettre en œuvre des stratégies telles que la redondance, la réplication et la détection des défaillances.
- Limites d'évolutivité : à mesure que le nombre total de nœuds de traitement augmente, la surcharge réseau associée augmente également. Chaque demande impliquant une communication sur le réseau peut augmenter le temps de latence et nuire aux avantages de la parallélisation.
- Problèmes de temps de réponse : pour les opérations qui nécessitent le traitement de tous les destinataires avant le traitement final, les performances de l'ensemble du système sont limitées par le temps de réponse du destinataire le plus lent.

- Réponses partielles : lorsque les demandes sont réparties entre plusieurs destinataires, le délai de réponse de certains destinataires peut être dépassé. Dans ces cas, l'implémentation doit indiquer au client que la réponse est incomplète. Vous pouvez également afficher les détails de l'agrégation des réponses à l'aide d'une interface utilisateur.
- Cohérence des données : lorsque vous traitez des données provenant de plusieurs destinataires, vous devez examiner attentivement les techniques de synchronisation des données et de résolution des conflits, afin de garantir l'exactitude et la cohérence des résultats agrégés finaux.

Mise en œuvre

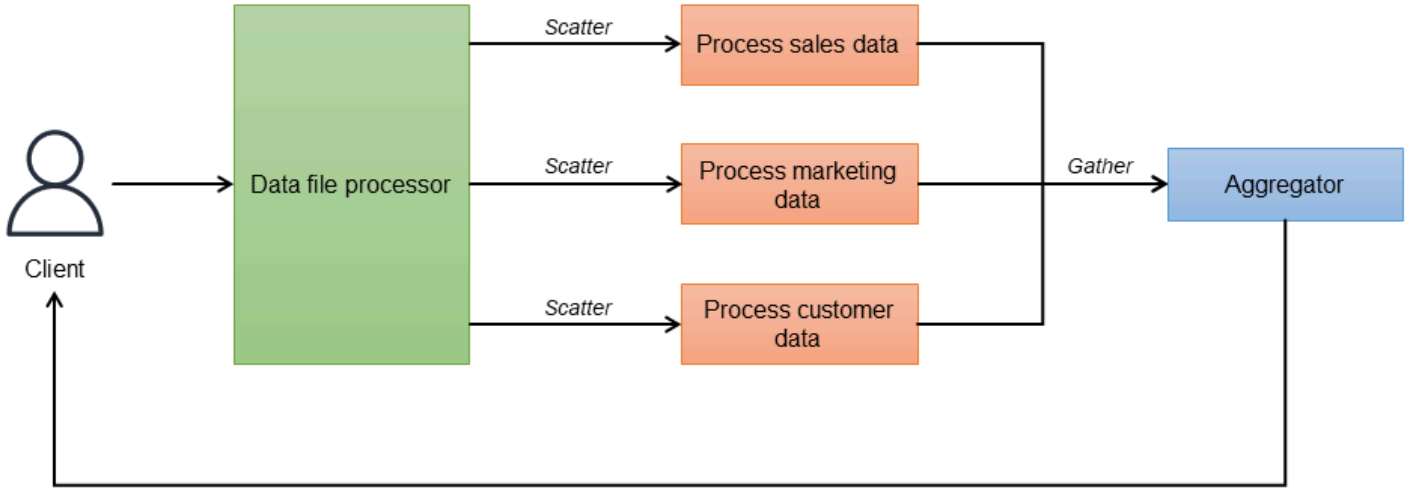
Architecture de haut niveau

Le modèle scatter-gather utilise un contrôleur racine pour distribuer les demandes aux destinataires qui les traiteront. Pendant la phase de diffusion, ce modèle peut utiliser deux mécanismes pour envoyer des messages aux destinataires :

- Répartition par distribution : l'application possède une liste connue de destinataires qui doivent être appelés pour obtenir les résultats. Les destinataires peuvent être différents processus dotés de fonctions uniques ou un processus unique qui a été étendu pour répartir la charge de traitement. Si l'un des nœuds de traitement arrive à expiration ou présente des retards de réponse, le contrôleur peut redistribuer le traitement à un autre nœud.
- Diffusion par enchère : l'application diffuse le message aux destinataires intéressés en utilisant un modèle de publication et d'abonnement. Dans ce cas, les destinataires peuvent s'abonner au message ou résilier leur abonnement à tout moment.

Dispersion par distribution

Dans la méthode de diffusion par distribution, le contrôleur racine divise la demande entrante en tâches indépendantes et les affecte aux destinataires disponibles (phase de diffusion). Chaque destinataire (processus, conteneur ou fonction Lambda) travaille indépendamment et en parallèle sur son calcul, et produit une partie de la réponse. Lorsque les destinataires terminent leurs tâches, ils envoient leurs réponses à un agrégateur (phase de collecte). L'agrégateur combine les réponses partielles et renvoie le résultat final au client. Le schéma suivant illustre ce flux de travail.



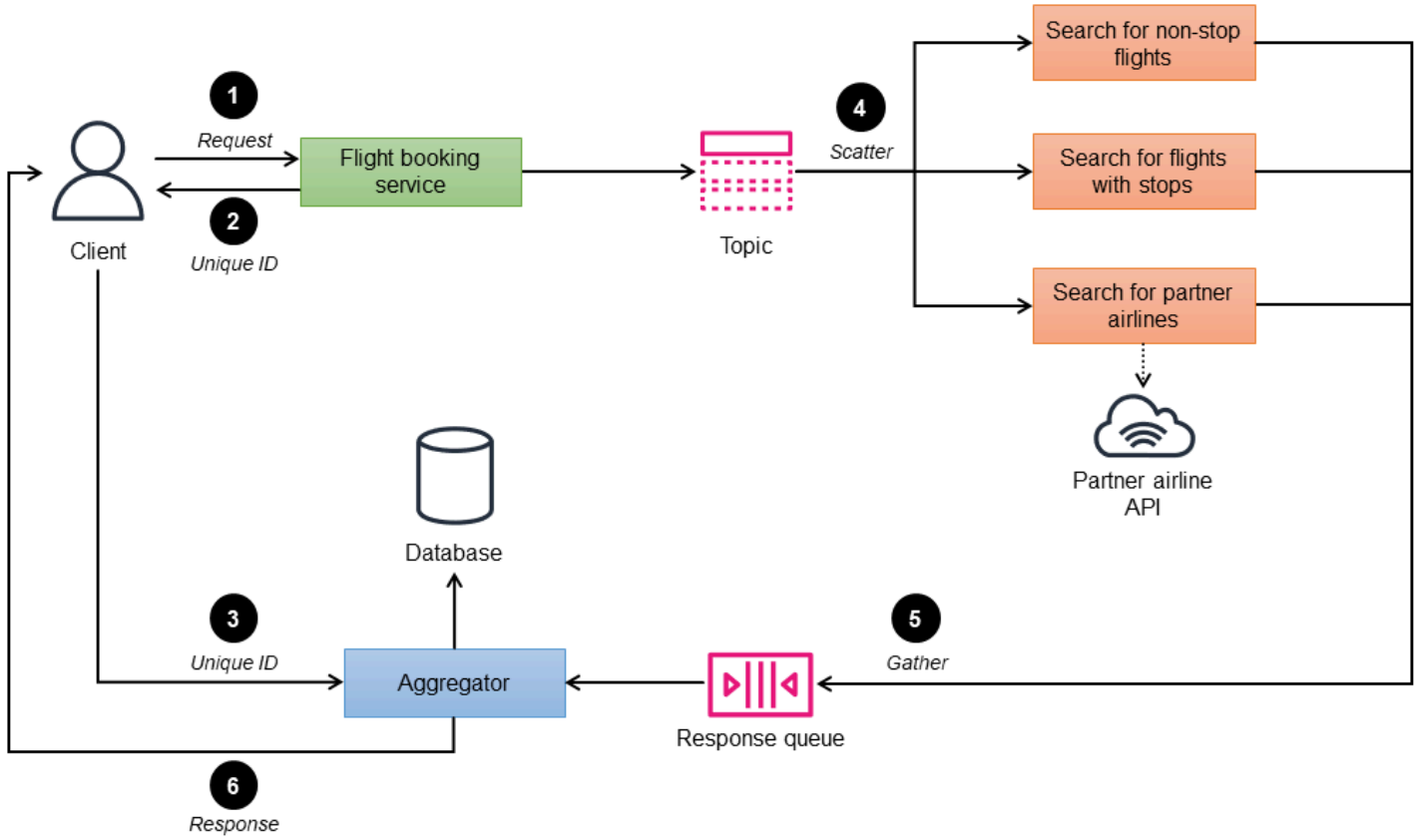
Le contrôleur (processeur de fichiers de données) orchestre l'ensemble des invocations et connaît tous les terminaux de réservation à appeler. Il peut configurer un paramètre de délai d'attente pour ignorer les réponses trop longues. Lorsque les demandes ont été envoyées, l'agrégateur attend les réponses de chaque point de terminaison. Pour mettre en œuvre la résilience, chaque microservice peut être déployé avec plusieurs instances pour l'équilibrage de charge. L'agrégateur obtient les résultats, les combine en un seul message de réponse et supprime les données dupliquées avant de poursuivre le traitement. Les réponses dont le délai est expiré sont ignorées. Le contrôleur peut également agir en tant qu'agrégateur au lieu d'utiliser un service d'agrégation distinct.

Dispersez par enchère

Si le contrôleur ne connaît pas les destinataires ou si les destinataires sont mal couplés, vous pouvez utiliser la méthode de diffusion par enchère. Dans cette méthode, les destinataires s'abonnent à un sujet et le contrôleur publie la demande dans le sujet. Les destinataires publient les résultats dans une file de réponses. Comme le contrôleur racine ne connaît pas les destinataires, le processus de collecte utilise un agrégateur (un autre modèle de messagerie) pour collecter les réponses et les distiller en un seul message de réponse. L'agrégateur utilise un identifiant unique pour identifier un groupe de demandes.

Par exemple, dans le schéma suivant, la méthode de diffusion par enchères est utilisée pour mettre en œuvre un service de réservation de vols pour le site Web d'une compagnie aérienne. Le site Web permet aux utilisateurs de rechercher et d'afficher les vols du transporteur de la compagnie aérienne et des transporteurs de ses partenaires, et doit afficher le statut de la recherche en temps réel. Le service de réservation de vols comprend trois microservices de recherche : les vols sans escale, les

vols avec escale et les compagnies aériennes partenaires. La recherche de compagnies aériennes partenaires appelle les points de terminaison de l'API du partenaire pour obtenir les réponses.

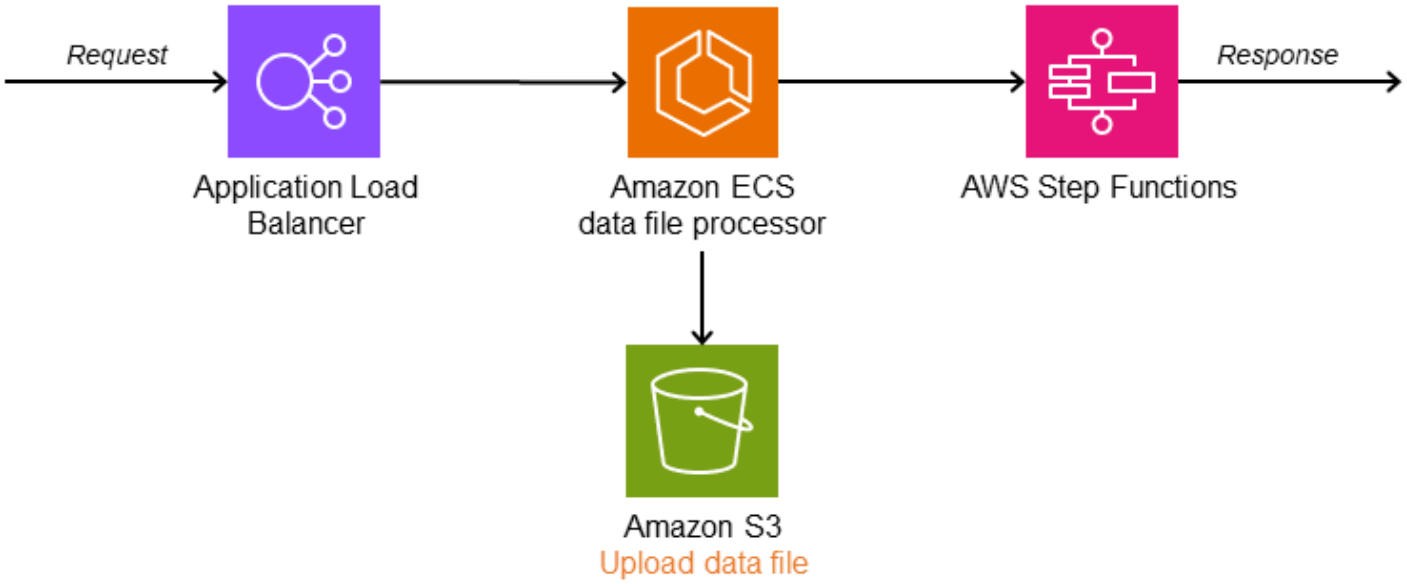


1. Le service de réservation de vols (contrôleur) prend les critères de recherche comme entrées par le client, traite et publie la demande sur le sujet.
2. Le contrôleur utilise un identifiant unique pour identifier chaque groupe de demandes.
3. Le client envoie l'identifiant unique à l'agrégateur pour l'étape 6.
4. Les microservices de recherche de réservation qui se sont abonnés au sujet de réservation reçoivent la demande.
5. Les microservices traitent la demande et renvoient la disponibilité des places pour les critères de recherche donnés à une file de réponses.
6. L'agrégateur rassemble tous les messages de réponse stockés dans une base de données temporaire, regroupe les vols par identifiant unique, crée une réponse unifiée unique et la renvoie au client.

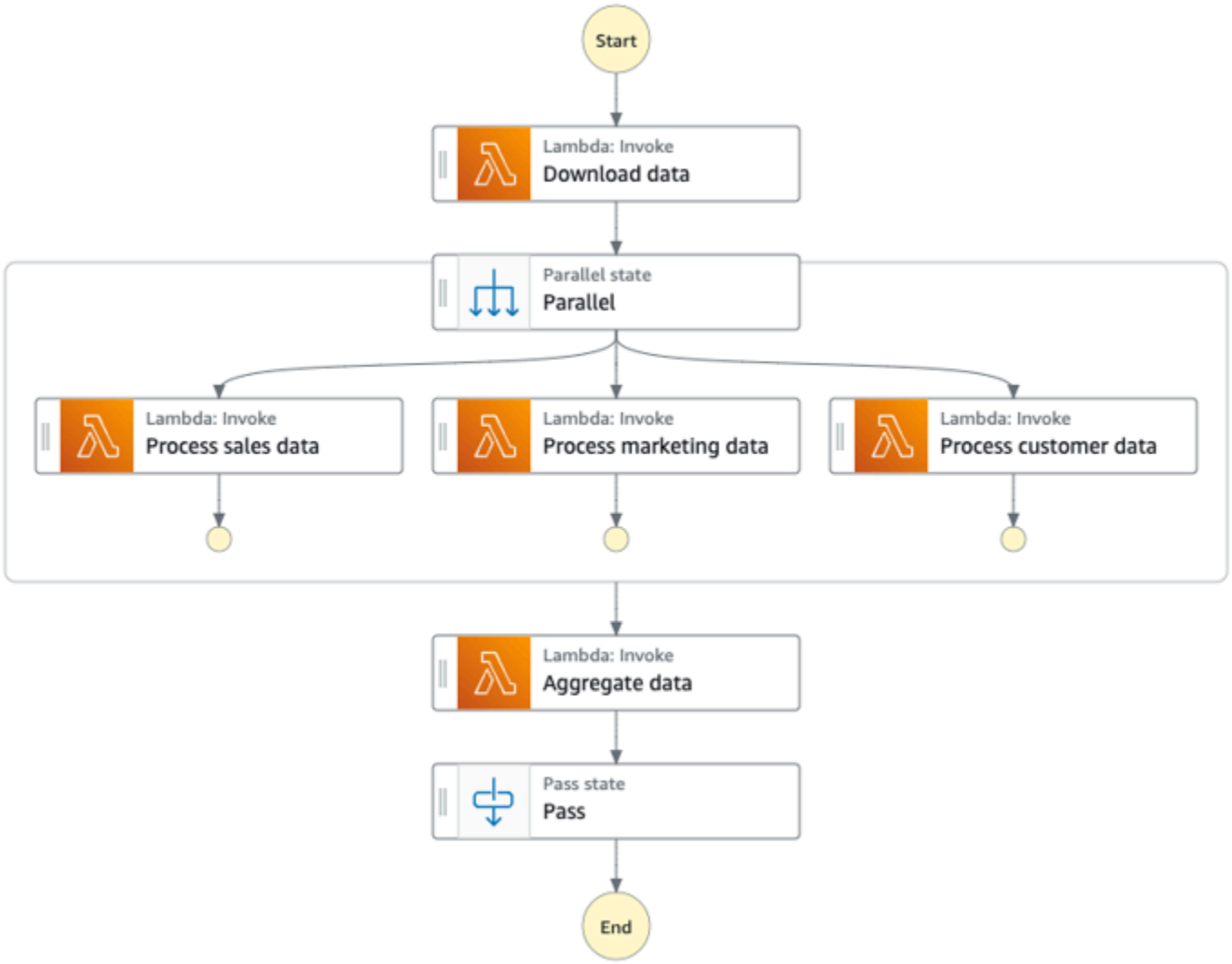
Mise en œuvre en utilisant Services AWS

Dispersion par distribution

Dans l'architecture suivante, le contrôleur racine est un processeur de fichiers de données (Amazon ECS) qui divise les données des demandes entrantes en compartiments Amazon Simple Storage Service (Amazon S3) individuels et lance un flux de travail. AWS Step Functions Le flux de travail télécharge les données et lance le traitement parallèle des fichiers. L'`Parallel`État attend que toutes les tâches renvoient une réponse. Une AWS Lambda fonction agrège les données et les enregistre à nouveau sur Amazon S3.

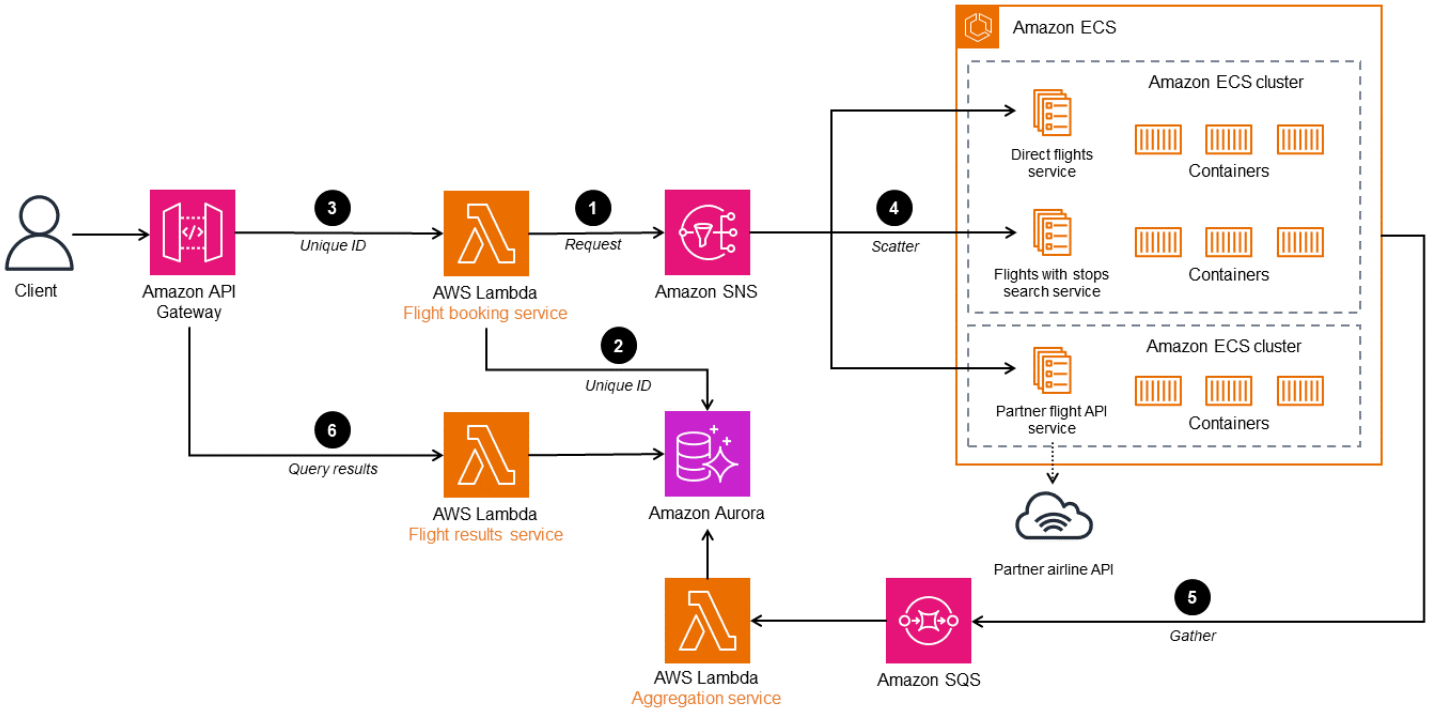


Le schéma suivant illustre le flux de travail Step Functions avec l'`Parallel`état.



Dispersez par enchère

Le schéma suivant montre AWS l'architecture de la méthode de diffusion par enchère. Le service de réservation de vol du contrôleur racine répartit la demande de recherche de vol entre plusieurs microservices. Un canal de publication et d'abonnement est mis en œuvre avec Amazon Simple Notification Service (Amazon SNS), qui est un service de messagerie géré pour les communications. Amazon SNS prend en charge les messages entre des applications de microservices découplées ou les communications directes avec les utilisateurs. Vous pouvez déployer les microservices des destinataires sur Amazon Elastic Kubernetes Service (Amazon EKS) ou Amazon Elastic Container Service (Amazon ECS) pour améliorer la gestion et l'évolutivité. Le service des résultats de vol renvoie les résultats au client. Il peut être implémenté dans AWS Lambda ou dans d'autres services d'orchestration de conteneurs tels qu'Amazon ECS ou Amazon EKS.



1. Le service de réservation de vols (contrôleur) prend les critères de recherche comme entrées par le client, traite et publie la demande sur la rubrique SNS.
2. Le contrôleur publie l'identifiant unique dans une base de données Amazon Aurora pour identifier la demande.
3. Le client envoie l'identifiant unique au client pour l'étape 6.
4. Les microservices de recherche de réservation qui se sont abonnés au sujet de réservation reçoivent la demande.
5. Les microservices traitent la demande et renvoient la disponibilité des places pour les critères de recherche donnés à une file de réponses dans Amazon Simple Queue Service (Amazon SQS). L'agrégateur rassemble tous les messages de réponse et les stocke dans une base de données temporaire.
6. Le service de résultats de vol regroupe les vols par identifiant unique, crée une réponse unifiée unique et la renvoie au client.

Si vous souhaitez ajouter une autre recherche de compagnies aériennes à cette architecture, vous devez ajouter un microservice qui s'abonne à la rubrique SNS et publie dans la file d'attente SQS.

En résumé, le modèle de collecte par dispersion permet aux systèmes distribués de réaliser une parallélisation efficace, de réduire le temps de latence et de gérer de manière fluide les communications asynchrones.

GitHub référentiel

Pour une implémentation complète de l'exemple d'architecture pour ce modèle, consultez le GitHub référentiel à l'adresse https://github.com/aws-samples/ asynchronous-messaging-workshop /tree/master/code/lab-3.

Ateliers

- Laboratoire de collecte de données dans le cadre de l'atelier sur les microservices découplés

Références du blog

- Modèles d'intégration des applications pour les microservices

Contenu connexe

- Modèle de publication et d'abonnement

Motif Strangler Fig

Intention

Le modèle Strangler Fig permet de migrer progressivement une application monolithique vers une architecture de microservices, tout en réduisant les risques de transformation et les interruptions d'activité.

Motivation

Les applications monolithiques sont développées pour fournir la plupart de leurs fonctionnalités dans un seul processus ou conteneur. Le code est étroitement lié. Par conséquent, les modifications apportées à l'application nécessitent de nouveaux tests approfondis afin d'éviter les problèmes de régression. Les modifications ne peuvent pas être testées isolément, ce qui a un impact sur le temps de cycle. Au fur et à mesure que l'application est enrichie de nouvelles fonctionnalités, une complexité élevée peut entraîner une augmentation du temps consacré à la maintenance, une augmentation du délai de mise sur le marché et, par conséquent, un ralentissement de l'innovation en matière de produits.

Lorsque la taille de l'application augmente, elle augmente la charge cognitive de l'équipe et peut entraîner des limites floues en matière de propriété de l'équipe. Il n'est pas possible de dimensionner les fonctionnalités individuelles en fonction de la charge : l'application entière doit être dimensionnée pour supporter les pics de charge. À mesure que les systèmes vieillissent, la technologie peut devenir obsolète, ce qui augmente les coûts de support. Les anciennes applications monolithiques suivent les meilleures pratiques disponibles au moment du développement et n'ont pas été conçues pour être distribuées.

Lorsqu'une application monolithique est migrée vers une architecture de microservices, elle peut être divisée en composants plus petits. Ces composants peuvent évoluer indépendamment, peuvent être publiés indépendamment et peuvent être détenus par des équipes individuelles. Cela se traduit par une vitesse de changement plus élevée, car les modifications sont localisées et peuvent être testées et publiées rapidement. Les modifications ont un impact moindre car les composants sont mal couplés et peuvent être déployés individuellement.

Remplacer complètement un monolithe par une application de microservices en réécrivant ou en refactorisant le code est une entreprise colossale et un gros risque. Une migration à grande échelle,

où le monolithe est migré en une seule opération, entraîne un risque de transformation et perturbe l'activité. Pendant le remaniement de l'application, il est extrêmement difficile, voire impossible, d'ajouter de nouvelles fonctionnalités.

Une façon de résoudre ce problème consiste à utiliser le motif Strangler Fig, introduit par Martin Fowler. Ce modèle implique de passer aux microservices en extrayant progressivement les fonctionnalités et en créant une nouvelle application autour du système existant. Les fonctionnalités du monolithe sont progressivement remplacées par des microservices, et les utilisateurs de l'application peuvent utiliser progressivement les fonctionnalités nouvellement migrées. Lorsque toutes les fonctionnalités sont transférées vers le nouveau système, l'application monolithique peut être mise hors service en toute sécurité.

Applicabilité

Utilisez le motif Strangler Fig lorsque :

- Vous souhaitez migrer progressivement votre application monolithique vers une architecture de microservices.
- Une approche de migration à grande échelle est risquée en raison de la taille et de la complexité du monolithe.
- L'entreprise souhaite ajouter de nouvelles fonctionnalités et est impatiente que la transformation soit terminée.
- Les utilisateurs finaux doivent être le moins impactés possible lors de la transformation.

Problèmes et considérations

- Accès à la base de code : pour implémenter le modèle Strangler Fig, vous devez avoir accès à la base de code de l'application Monolith. Au fur et à mesure que les fonctionnalités sont migrées hors du monolithe, vous devrez apporter des modifications mineures au code et implémenter une couche anticorruption au sein du monolithe pour acheminer les appels vers de nouveaux microservices. Vous ne pouvez pas intercepter les appels sans accès par code. L'accès à la base de code est également essentiel pour rediriger les demandes entrantes : une certaine refactorisation du code peut être nécessaire afin que la couche proxy puisse intercepter les appels aux fonctionnalités migrées et les acheminer vers des microservices.
- Domaine peu clair : La décomposition prématurée des systèmes peut s'avérer coûteuse, en particulier lorsque le domaine n'est pas clair, et il est possible de se tromper dans les limites des

services. La conception axée sur le domaine (DDD) est un mécanisme permettant de comprendre le domaine, et le storming d'événements est une technique permettant de déterminer les limites du domaine.

- Identification des microservices : vous pouvez utiliser DDD comme outil clé pour identifier les microservices. Pour identifier les microservices, recherchez les divisions naturelles entre les classes de services. De nombreux services posséderont leur propre objet d'accès aux données et se découpleront facilement. Les services qui ont une logique métier associée et les classes qui n'ont pas ou peu de dépendances sont de bons candidats pour les microservices. Vous pouvez refactoriser le code avant de décomposer le monolithe pour éviter un couplage serré. Vous devez également tenir compte des exigences de conformité, de la cadence de publication, de la situation géographique des équipes, des besoins en matière de mise à l'échelle, des besoins technologiques axés sur les cas d'utilisation et de la charge cognitive des équipes.
- Couche anticorruption : pendant le processus de migration, lorsque les fonctionnalités du monolithe doivent appeler les fonctionnalités migrées en tant que microservices, vous devez implémenter une couche anticorruption (ACL) qui achemine chaque appel vers le microservice approprié. Afin de découpler et d'empêcher toute modification des appelants existants au sein du monolithe, l'ACL fonctionne comme un adaptateur ou une façade qui convertit les appels vers la nouvelle interface. Ceci est décrit en détail dans la section Implémentation du modèle ACL plus haut dans ce guide.
- Défaillance de la couche proxy : pendant la migration, une couche proxy intercepte les demandes envoyées à l'application monolithique et les achemine vers le système existant ou le nouveau système. Cependant, cette couche proxy peut devenir un point de défaillance unique ou un goulot d'étranglement des performances.
- Complexité de l'application : les grands monolithes sont ceux qui tirent le meilleur parti du motif Strangler Fig. Pour les petites applications, où la complexité du refactoring complet est faible, il peut être plus efficace de réécrire l'application dans une architecture de microservices plutôt que de la migrer.
- Interactions entre les services : les microservices peuvent communiquer de manière synchrone ou asynchrone. Lorsqu'une communication synchrone est requise, déterminez si les délais d'attente peuvent entraîner une consommation de connexion ou de pool de threads, entraînant des problèmes de performances de l'application. Dans de tels cas, utilisez le schéma du disjoncteur pour annuler immédiatement une panne pour les opérations susceptibles d'échouer pendant de longues périodes. La communication asynchrone peut être réalisée à l'aide d'événements et de files d'attente de messagerie.
- Agrégation de données : dans une architecture de microservices, les données sont réparties entre les bases de données. Lorsque l'agrégation des données est requise, vous pouvez utiliser

AWS AppSyncle modèle frontal ou le modèle de ségrégation des responsabilités des requêtes de commande (CQRS) dans le backend.

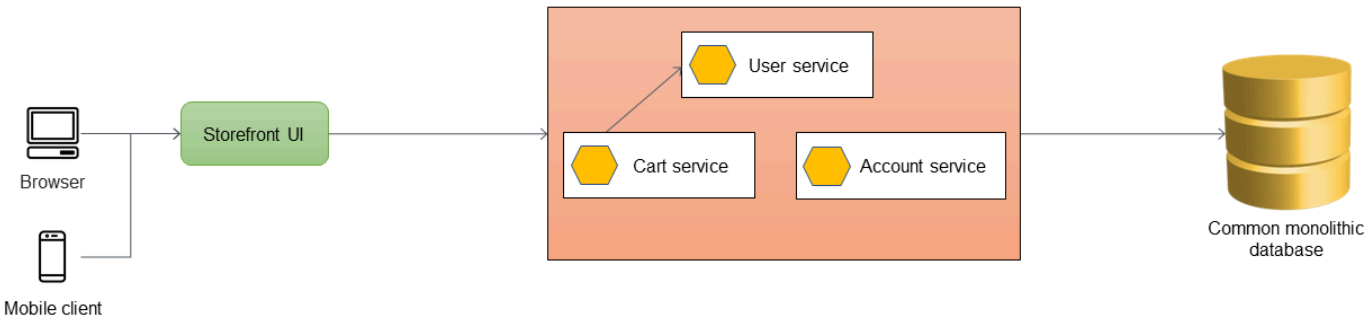
- Cohérence des données : les microservices sont propriétaires de leur magasin de données, et l'application monolithique peut également potentiellement utiliser ces données. Pour activer le partage, vous pouvez synchroniser le magasin de données des nouveaux microservices avec la base de données de l'application monolithique à l'aide d'une file d'attente et d'un agent. Cela peut toutefois entraîner une redondance des données et une éventuelle cohérence entre deux magasins de données. Nous vous recommandons donc de le traiter comme une solution tactique jusqu'à ce que vous puissiez établir une solution à long terme, telle qu'un lac de données.

Mise en œuvre

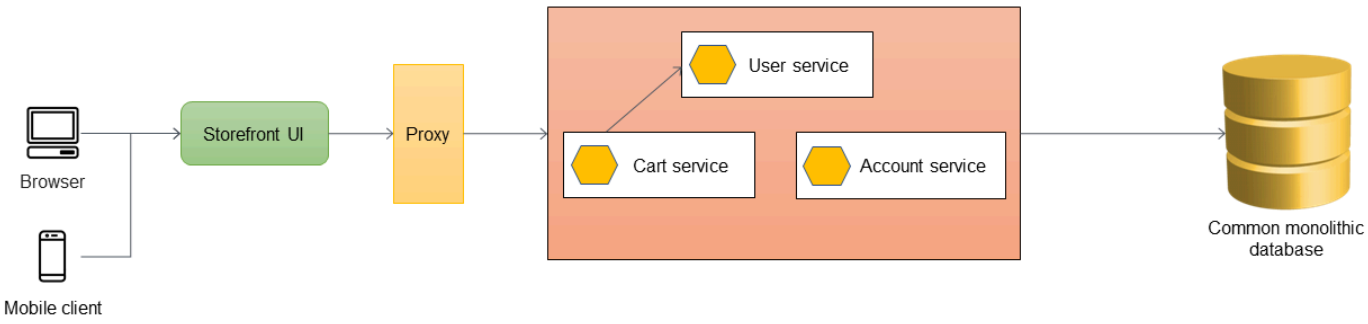
Dans le modèle Strangler Fig, vous remplacez une fonctionnalité spécifique par un nouveau service ou une nouvelle application, un composant à la fois. Une couche proxy intercepte les demandes envoyées à l'application monolithique et les achemine vers le système existant ou le nouveau système. Comme la couche proxy dirige les utilisateurs vers l'application appropriée, vous pouvez ajouter des fonctionnalités au nouveau système tout en veillant à ce que le monolithe continue de fonctionner. Le nouveau système remplace finalement toutes les fonctionnalités de l'ancien système, et vous pouvez le mettre hors service.

Architecture de haut niveau

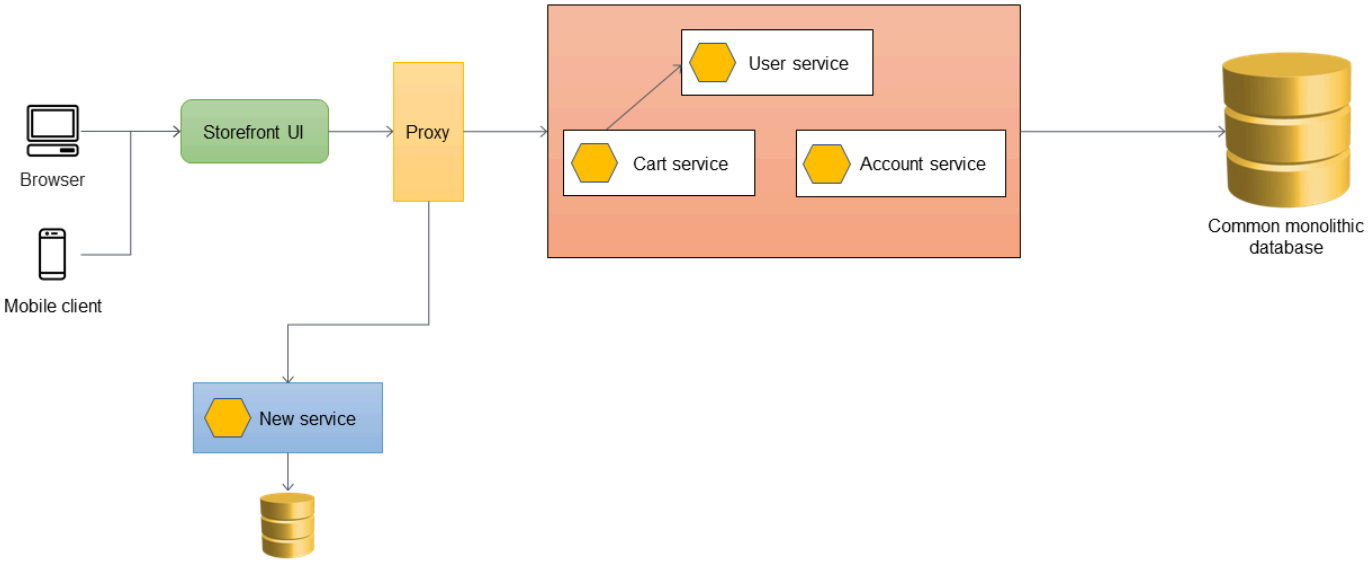
Dans le schéma suivant, une application monolithique comporte trois services : le service utilisateur, le service de panier et le service de compte. Le service de panier dépend du service utilisateur, et l'application utilise une base de données relationnelle monolithique.



La première étape consiste à ajouter une couche proxy entre l'interface utilisateur de Storefront et l'application monolithique. Au début, le proxy achemine tout le trafic vers l'application monolithique.

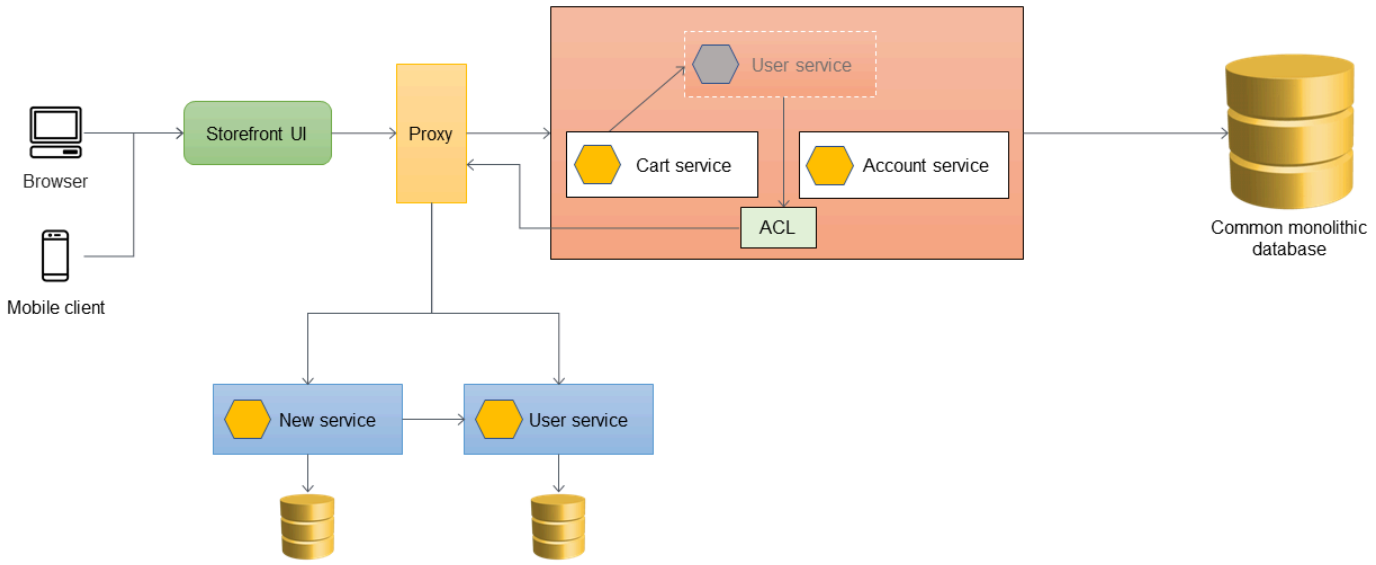


Lorsque vous souhaitez ajouter de nouvelles fonctionnalités à votre application, vous les implémentez sous forme de nouveaux microservices au lieu d'ajouter des fonctionnalités au monolithe existant. Cependant, vous continuez à corriger les bogues du monolithe pour garantir la stabilité de l'application. Dans le schéma suivant, la couche proxy achemine les appels vers le monolithe ou vers le nouveau microservice en fonction de l'URL de l'API.



Ajouter une couche anticorruption

Dans l'architecture suivante, le service utilisateur a été migré vers un microservice. Le service de panier appelle le service utilisateur, mais l'implémentation n'est plus disponible dans le monolithe. En outre, l'interface du service nouvellement migré peut ne pas correspondre à son interface précédente dans l'application monolithique. Pour faire face à ces changements, vous devez implémenter une ACL. Au cours du processus de migration, lorsque les fonctionnalités du monolithe doivent appeler les fonctionnalités qui ont été migrées en tant que microservices, l'ACL convertit les appels vers la nouvelle interface et les achemine vers le microservice approprié.



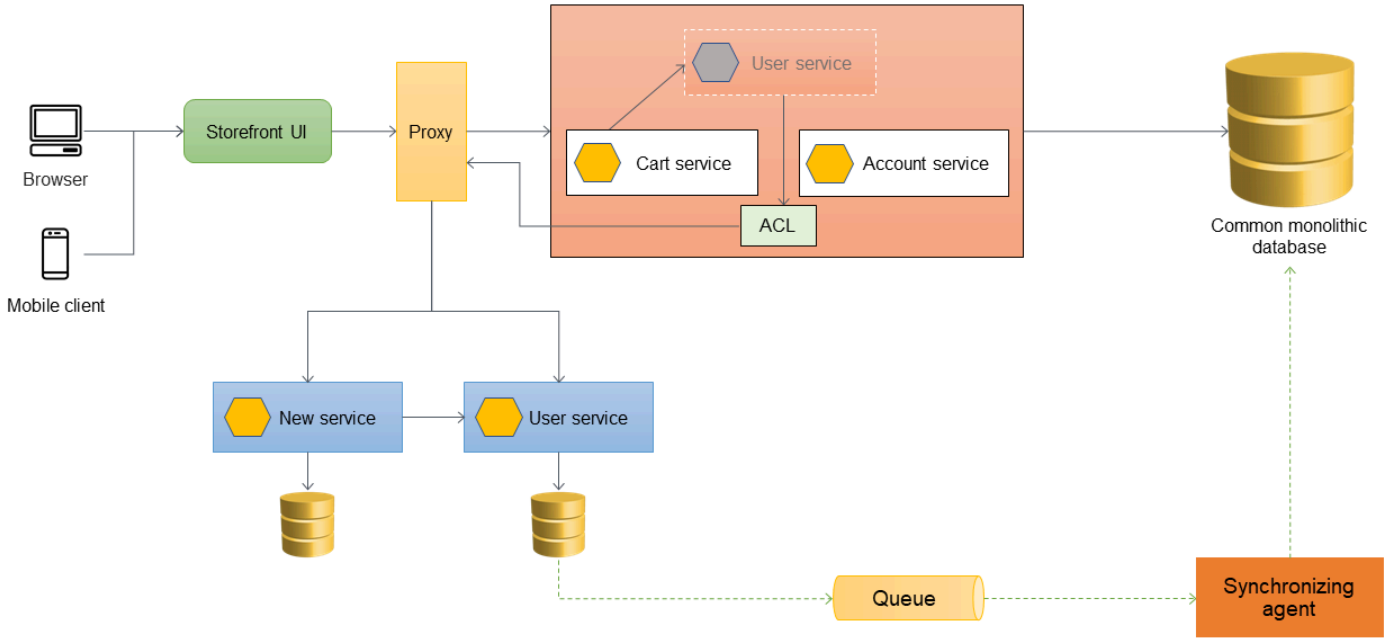
Vous pouvez implémenter l'ACL dans l'application monolithique sous la forme d'une classe spécifique au service qui a été migré ; par exemple, `UserServiceFacade` ou. `UserServiceAdapter` L'ACL doit être mise hors service une fois que tous les services dépendants ont été migrés vers l'architecture des microservices.

Lorsque vous utilisez l'ACL, le service cart appelle toujours le service utilisateur au sein du monolithe, et le service utilisateur redirige l'appel vers le microservice via l'ACL. Le service de panier doit toujours appeler le service utilisateur sans être au courant de la migration du microservice. Ce couplage souple est nécessaire pour réduire la régression et les perturbations des activités.

Gestion de la synchronisation des données

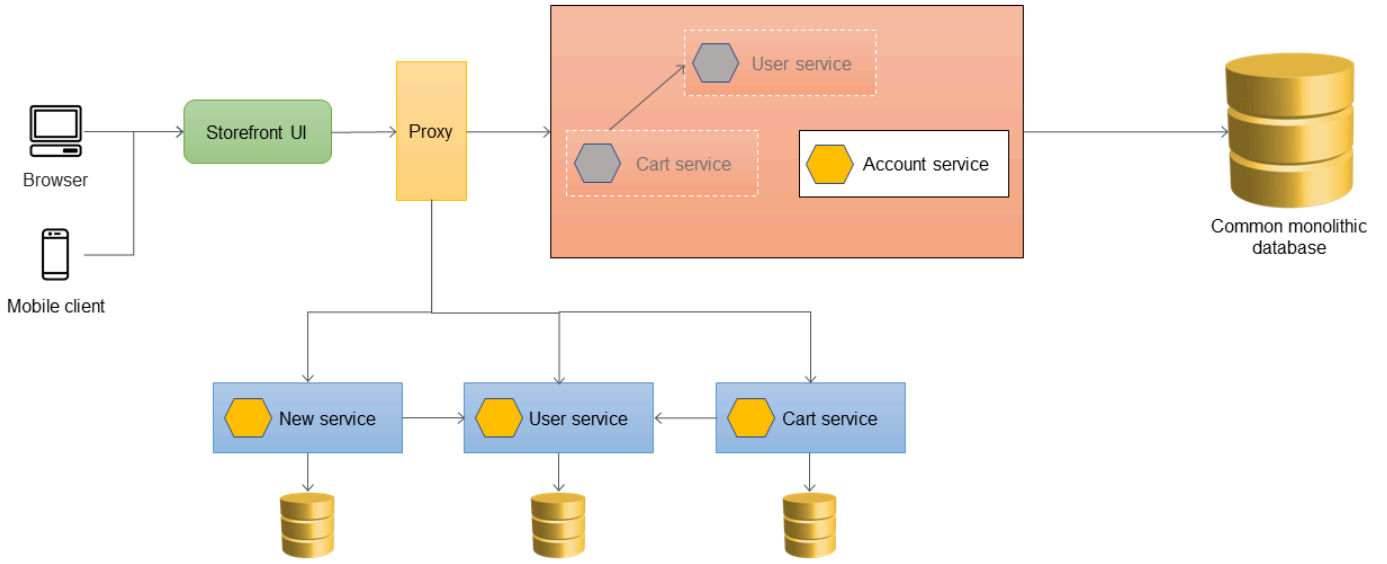
La meilleure pratique consiste à ce que le microservice soit propriétaire de ses données. Le service utilisateur stocke ses données dans son propre magasin de données. Il peut être nécessaire de synchroniser les données avec la base de données monolithique pour gérer les dépendances telles que les rapports et pour prendre en charge les applications en aval qui ne sont pas encore prêtes à accéder directement aux microservices. L'application monolithique peut également avoir besoin des données pour d'autres fonctions et composants qui n'ont pas encore été migrés vers des microservices. La synchronisation des données est donc nécessaire entre le nouveau microservice et le monolithe. Pour synchroniser les données, vous pouvez introduire un agent de synchronisation entre le microservice utilisateur et la base de données monolithique, comme indiqué dans le schéma suivant. Le microservice utilisateur envoie un événement à la file d'attente chaque fois que sa base de données est mise à jour. L'agent de synchronisation écoute la file d'attente et met continuellement

à jour la base de données monolithique. Les données de la base de données monolithique sont finalement cohérentes avec les données en cours de synchronisation.

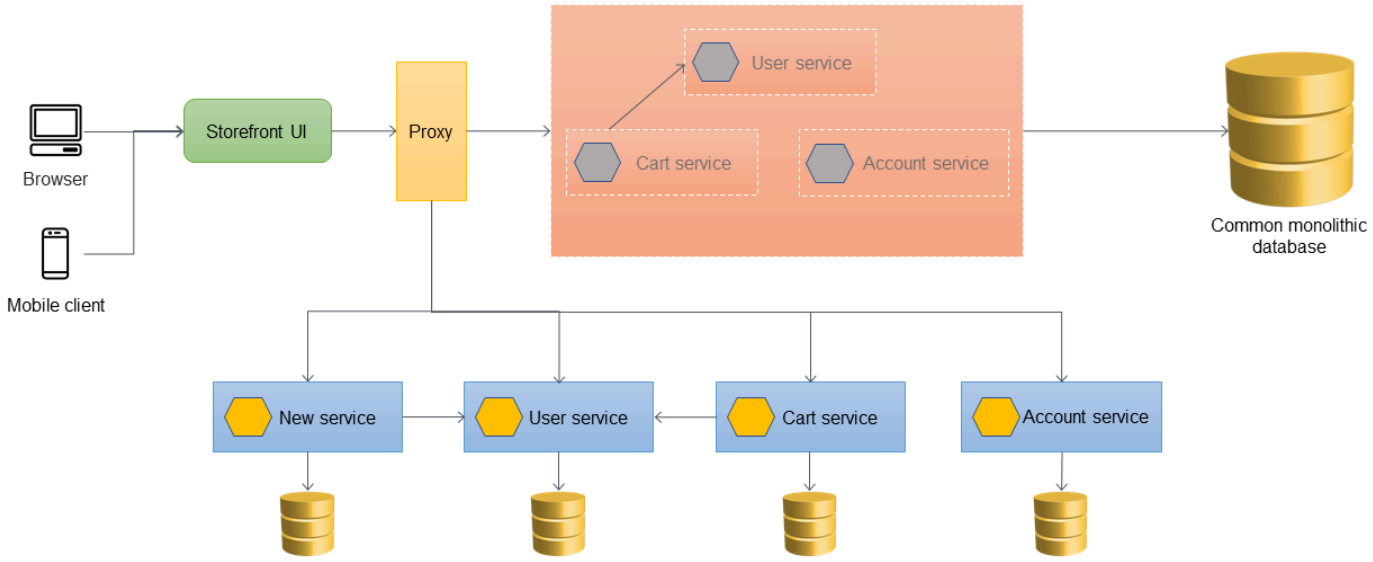


Migration de services supplémentaires

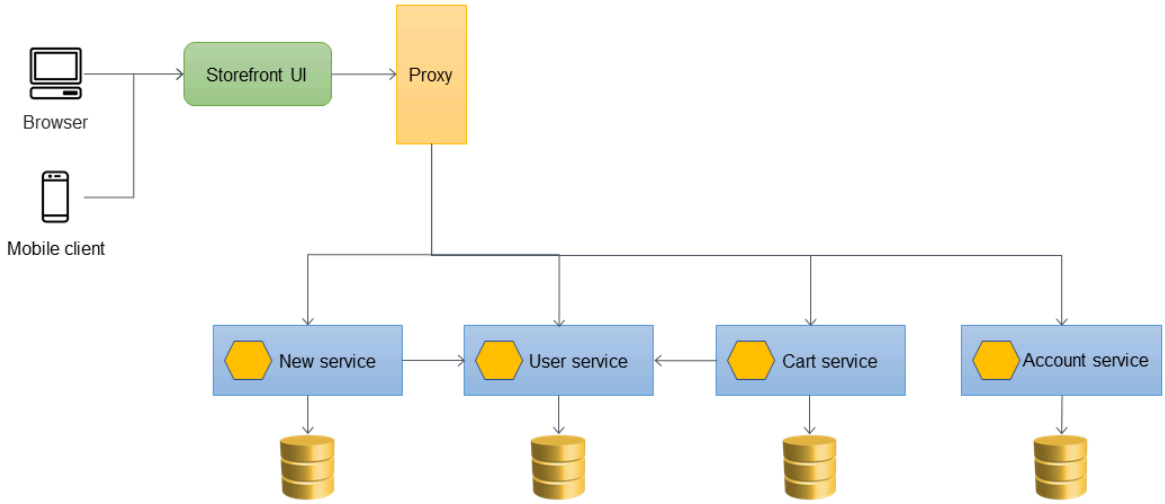
Lorsque le service de panier est migré hors de l'application monolithique, son code est révisé pour appeler directement le nouveau service, de sorte que l'ACL n'achemine plus ces appels. Le schéma suivant illustre cette architecture.



Le schéma suivant montre l'état d'étranglement final où tous les services ont été migrés hors du monolithe et où il ne reste que le squelette du monolithe. Les données historiques peuvent être migrées vers des magasins de données appartenant à des services individuels. L'ACL peut être retiré et le monolithe est prêt à être mis hors service à ce stade.



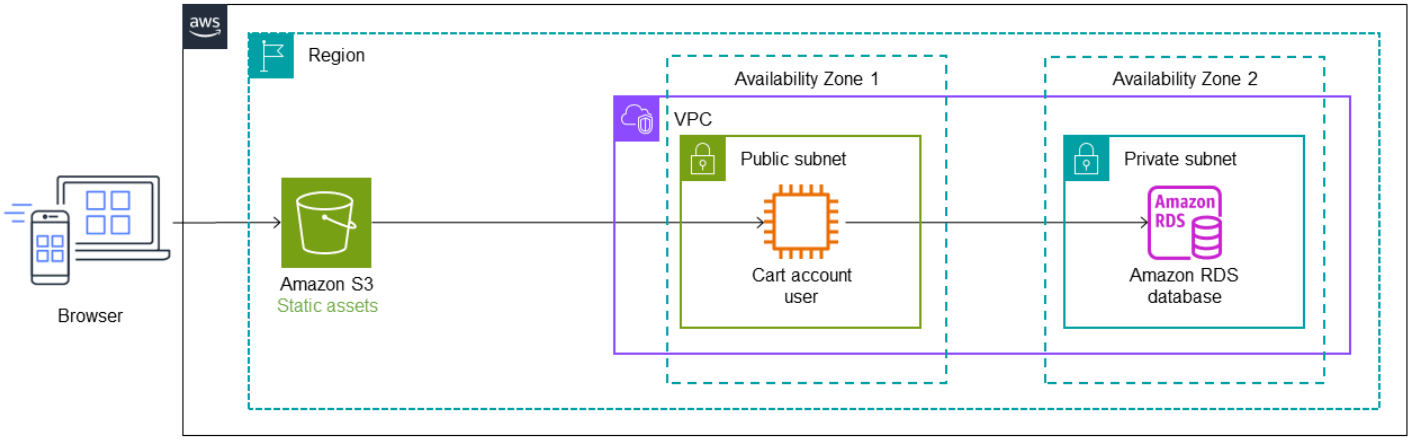
Le schéma suivant montre l'architecture finale après la mise hors service de l'application monolithique. Vous pouvez héberger les microservices individuels via une URL basée sur les ressources (telle que`http://www.storefront.com/user`) ou via leur propre domaine (par exemple,`http://user.storefront.com`) en fonction des exigences de votre application. Pour plus d'informations sur les principales méthodes permettant d'exposer les API HTTP aux consommateurs en amont à l'aide de noms d'hôtes et de chemins, consultez la section sur les modèles de routage des API.



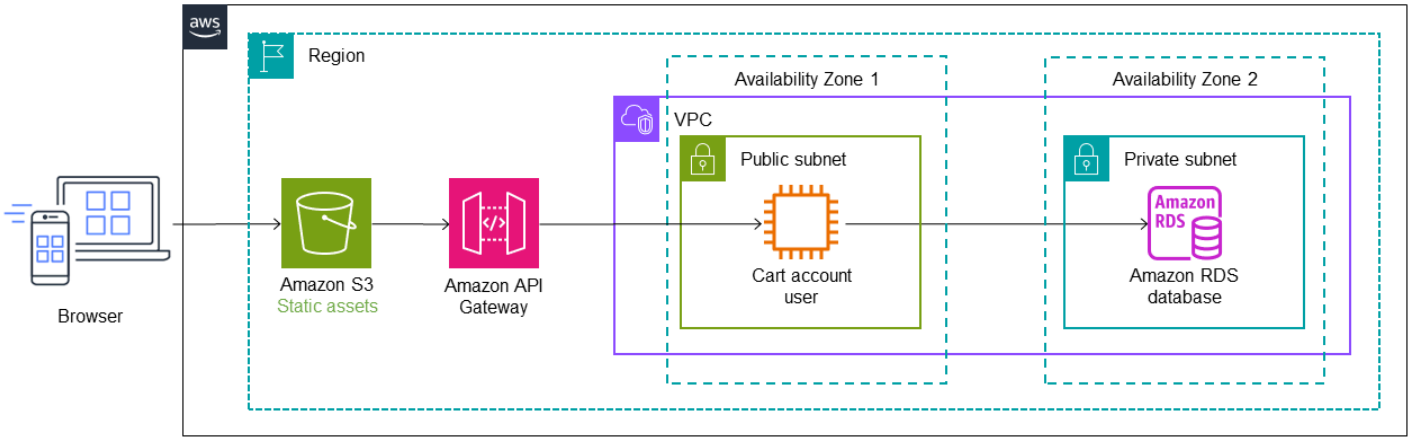
Mise en œuvre au moyen AWS de services

Utilisation d'API Gateway comme proxy d'application

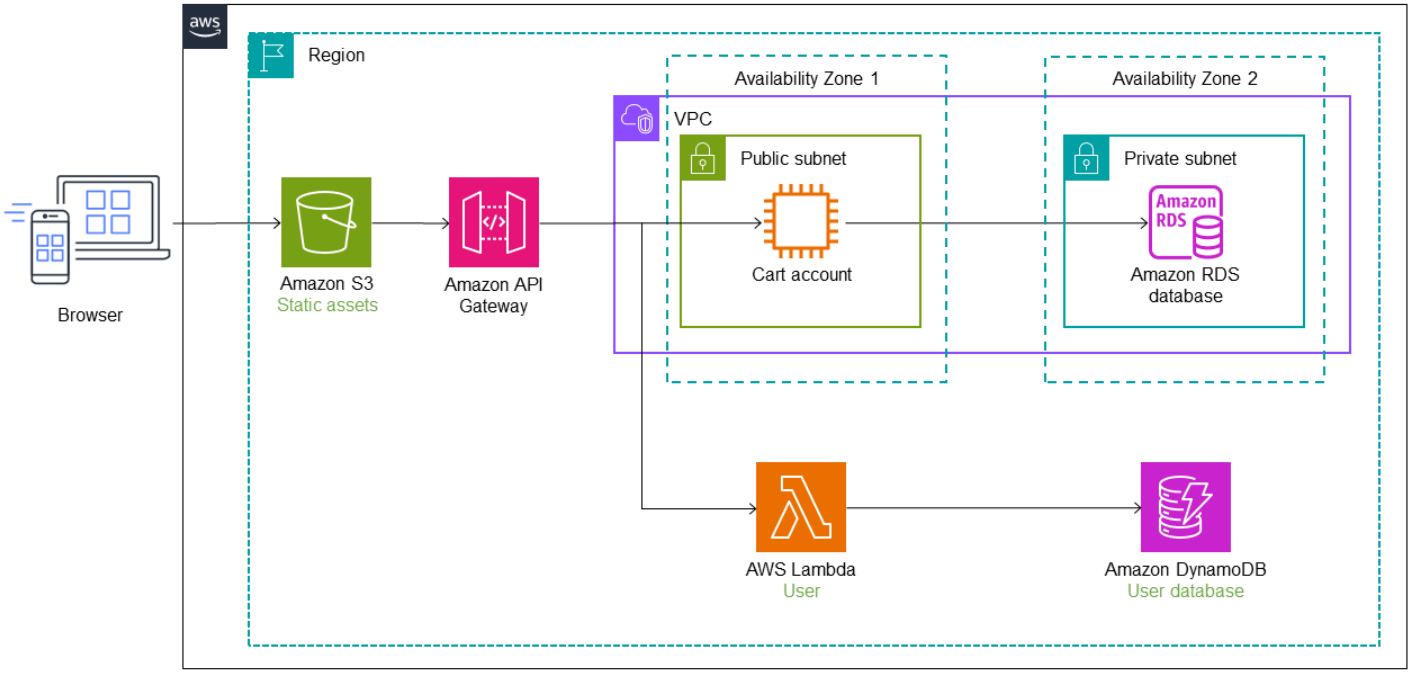
Le schéma suivant montre l'état initial de l'application monolithique. Supposons qu'il ait été AWS migré à l'aide d'une lift-and-shift stratégie. Il s'exécute donc sur une instance Amazon Elastic Compute Cloud (Amazon EC2) et utilise une base de données Amazon Relational Database Service (Amazon RDS). Pour des raisons de simplicité, l'architecture utilise un seul cloud privé virtuel (VPC) avec un sous-réseau privé et un sous-réseau public, et supposons que les microservices seront initialement déployés au sein de ce même cloud. Compte AWS (La meilleure pratique dans les environnements de production consiste à utiliser une architecture multi-comptes pour garantir l'indépendance du déploiement.) L'instance EC2 réside dans une seule zone de disponibilité du sous-réseau public, et l'instance RDS réside dans une seule zone de disponibilité du sous-réseau privé. Amazon Simple Storage Service (Amazon S3) stocke les actifs statiques tels que JavaScript les fichiers CSS et React pour le site Web.



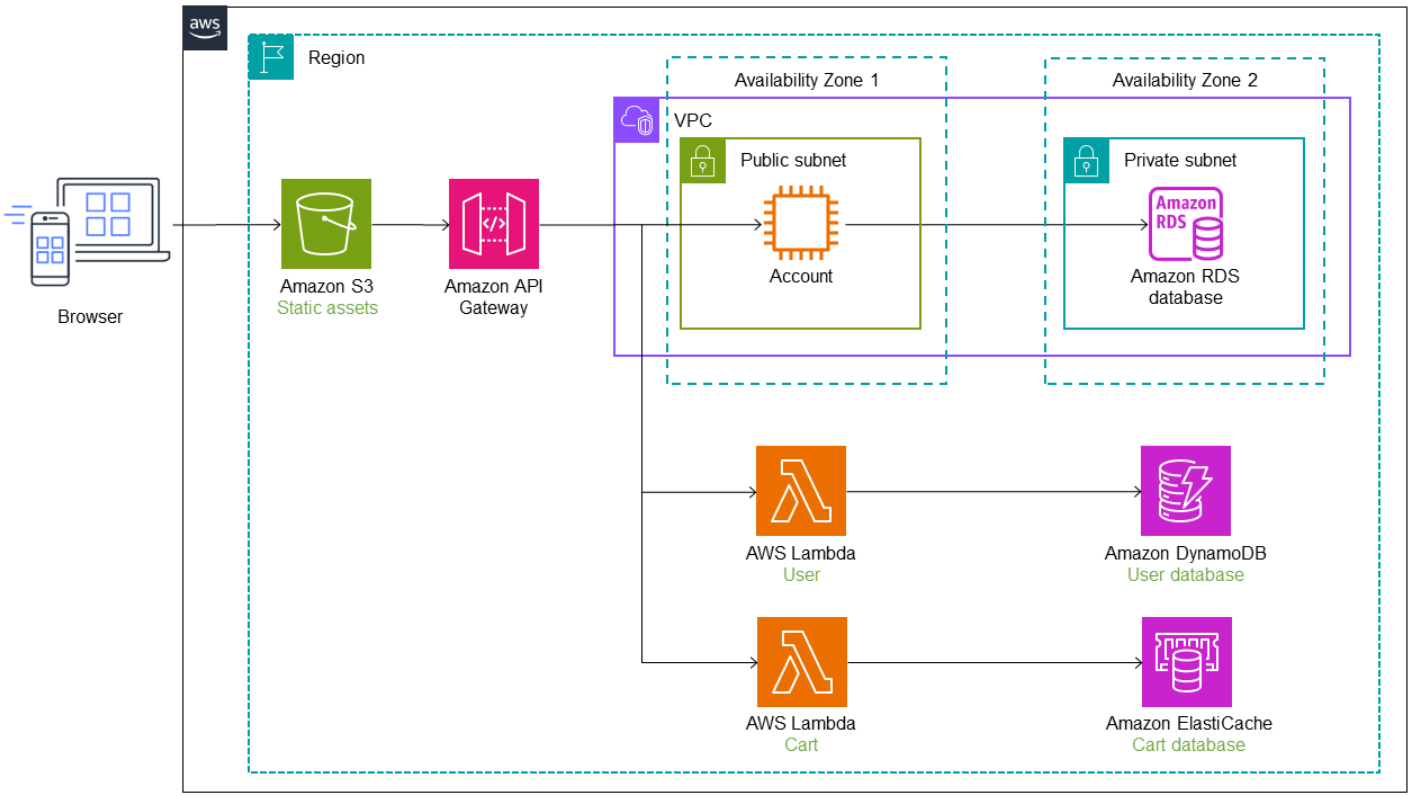
Dans l'architecture suivante, AWS Migration Hub Refactor Spacesdéploie Amazon API Gateway devant l'application monolithique. Refactor Spaces crée une infrastructure de refactorisation au sein de votre compte, et API Gateway fait office de couche proxy pour acheminer les appels vers le monolithe. Dans un premier temps, tous les appels sont acheminés vers l'application monolithique via la couche proxy. Comme indiqué précédemment, les couches proxy peuvent devenir un point de défaillance unique. Cependant, l'utilisation d'API Gateway comme proxy limite le risque car il s'agit d'un service multi-AZ sans serveur.



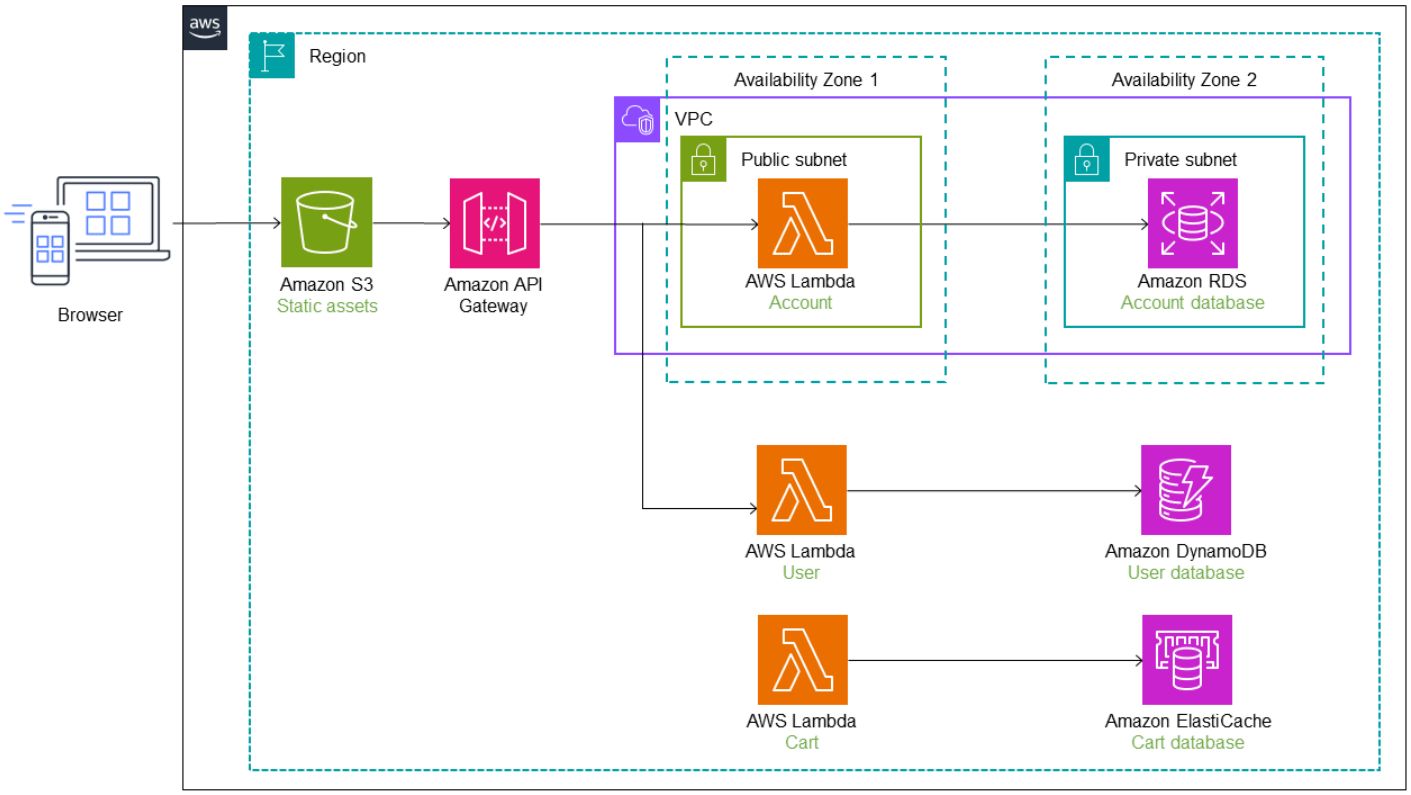
Le service utilisateur est migré vers une fonction Lambda, et une base de données Amazon DynamoDB stocke ses données. Un point de terminaison de service Lambda et une route par défaut sont ajoutés à Refactor Spaces, et API Gateway est automatiquement configuré pour acheminer les appels vers la fonction Lambda. Pour plus de détails sur la mise en œuvre, consultez le module 2 de l'atelier de modernisation itérative des applications.



Dans le schéma suivant, le service cart a également été migré du monolithe vers une fonction Lambda. Une route et un point de terminaison de service supplémentaires sont ajoutés à Refactor Spaces, et le trafic est automatiquement transféré à la fonction `Cart` Lambda. Le magasin de données pour la fonction Lambda est géré par Amazon. ElastiCache L'application monolithique est toujours présente dans l'instance EC2 avec la base de données Amazon RDS.



Dans le schéma suivant, le dernier service (compte) est migré du monolithe vers une fonction Lambda. Il continue d'utiliser la base de données Amazon RDS d'origine. La nouvelle architecture comporte désormais trois microservices dotés de bases de données distinctes. Chaque service utilise un type de base de données différent. Ce concept d'utilisation de bases de données spécialement conçues pour répondre aux besoins spécifiques des microservices est appelé persistance polyglotte. Les fonctions Lambda peuvent également être implémentées dans différents langages de programmation, selon le cas d'utilisation. Lors du refactoring, Refactor Spaces automatise le transfert et le routage du trafic vers Lambda. Cela permet à vos concepteurs de gagner du temps pour concevoir, déployer et configurer l'infrastructure de routage.



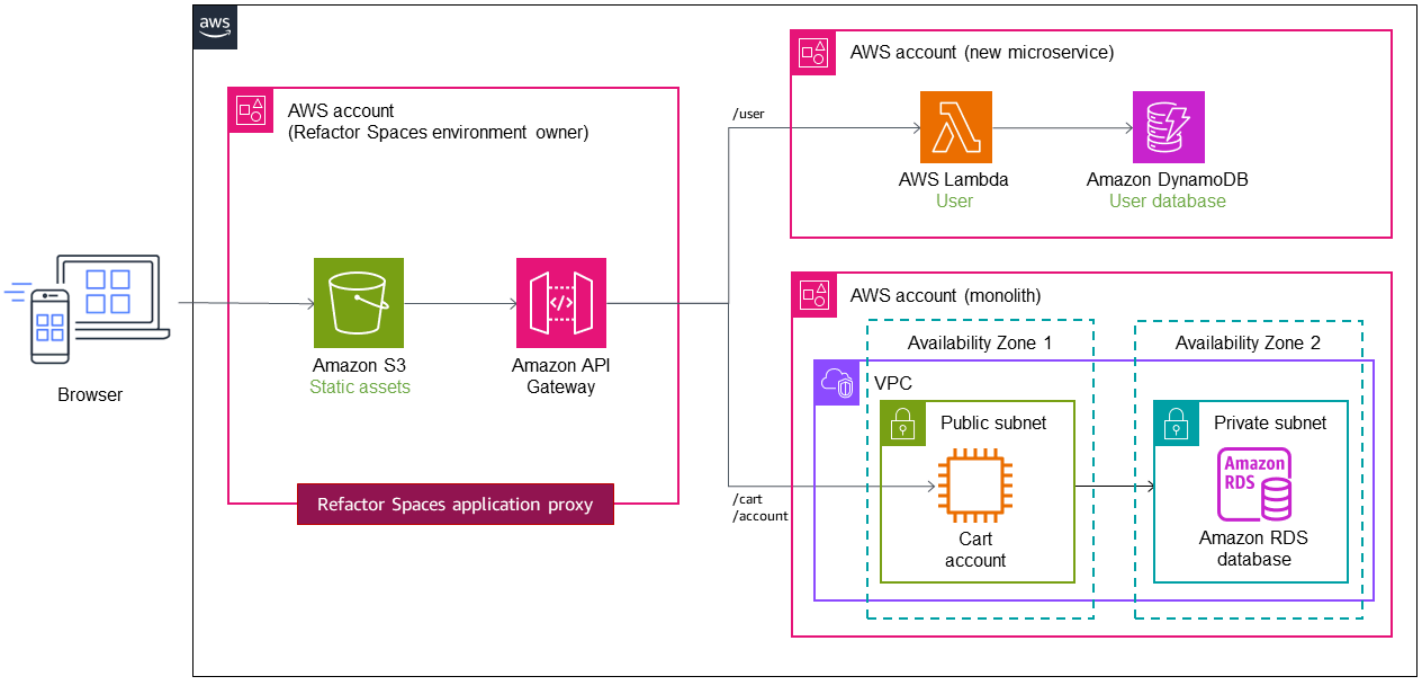
Utilisation de plusieurs comptes

Dans l'implémentation précédente, nous utilisions un seul VPC doté d'un sous-réseau privé et d'un sous-réseau public pour l'application monolithique, et nous avons déployé les microservices au sein de celui-ci dans un souci de simplicitéCompte AWS. Cependant, c'est rarement le cas dans les scénarios réels, où les microservices sont souvent déployés en plusieurs Comptes AWS pour garantir l'indépendance du déploiement. Dans une structure multi-comptes, vous devez configurer le routage du trafic du monolithe vers les nouveaux services dans différents comptes.

Refactor Spaces vous aide à créer et à configurer l'AWSinfrastructure pour le routage des appels d'API en dehors de l'application monolithique. Refactor Spaces orchestre les politiques API Gateway, Network Load Balancer et AWS Identity and Access Managementbasées sur les ressources (IAM) au sein de AWS vos comptes dans le cadre de ses ressources applicatives. Vous pouvez ajouter de manière transparente de nouveaux services dans un seul Compte AWS ou plusieurs comptes à un point de terminaison HTTP externe. Toutes ces ressources sont orchestrées à l'intérieur de votre ordinateur Compte AWS et peuvent être personnalisées et configurées après le déploiement.

Supposons que les services utilisateur et panier soient déployés sur deux comptes différents, comme le montre le schéma suivant. Lorsque vous utilisez Refactor Spaces, il vous suffit de configurer le point de terminaison du service et l'itinéraire. Refactor Spaces automatise l'intégration entre API

Gateway et Lambda et la création de politiques de ressources Lambda, afin que vous puissiez vous concentrer sur la refactorisation sécurisée des services hors du monolithe.



Pour un didacticiel vidéo sur l'utilisation des espaces de refactorisation, voir Refactoriser les applications de manière incrémentielle avec. AWS Migration Hub Refactor Spaces

Atelier

- Atelier de modernisation itérative des applications

Références du blog

- AWS Migration Hub Refactor Spaces
- Plongez en profondeur sur un AWS Migration Hub Refactor Spaces
- Architecture de référence des pipelines de déploiement et implémentations de référence

Contenu connexe

- Modèles de routage des API
- Documentation sur Refactor Spaces

Modèle de boîte d’envoi transactionnelle

Intention

Le modèle de boîte d’envoi transactionnelle résout le problème des opérations d’écriture double qui se produit dans les systèmes distribués lorsqu’une seule opération implique à la fois une opération d’écriture dans la base de données et une notification de message ou d’événement. Une opération d’écriture double se produit lorsqu’une application écrit sur deux systèmes différents. Par exemple, lorsqu’un microservice doit conserver des données dans la base de données et envoyer un message pour avertir les autres systèmes. L’échec de l’une de ces opérations peut entraîner des données incohérentes.

Motivation

Lorsqu’un microservice envoie une notification d’événement après une mise à jour de base de données, ces deux opérations doivent être exécutées de manière atomique pour garantir la fiabilité et la cohérence des données.

- Si la mise à jour de la base de données est réussie, mais que la notification d’événement échoue, le service en aval ne sera pas au courant de la modification et le système peut entrer dans un état incohérent.
- Si la mise à jour de la base de données échoue, mais que la notification d’événement est envoyée, les données risquent d’être corrompues, ce qui peut affecter la fiabilité du système.

Applicabilité

Utilisez le modèle de boîte d’envoi transactionnelle lorsque :

- Vous créez une application pilotée par des événements dans laquelle une mise à jour de base de données déclenche une notification d’événement.
- Vous souhaitez garantir l’atomicité dans les opérations impliquant deux services.
- Vous souhaitez implémenter le modèle d’approvisionnement d’événement.

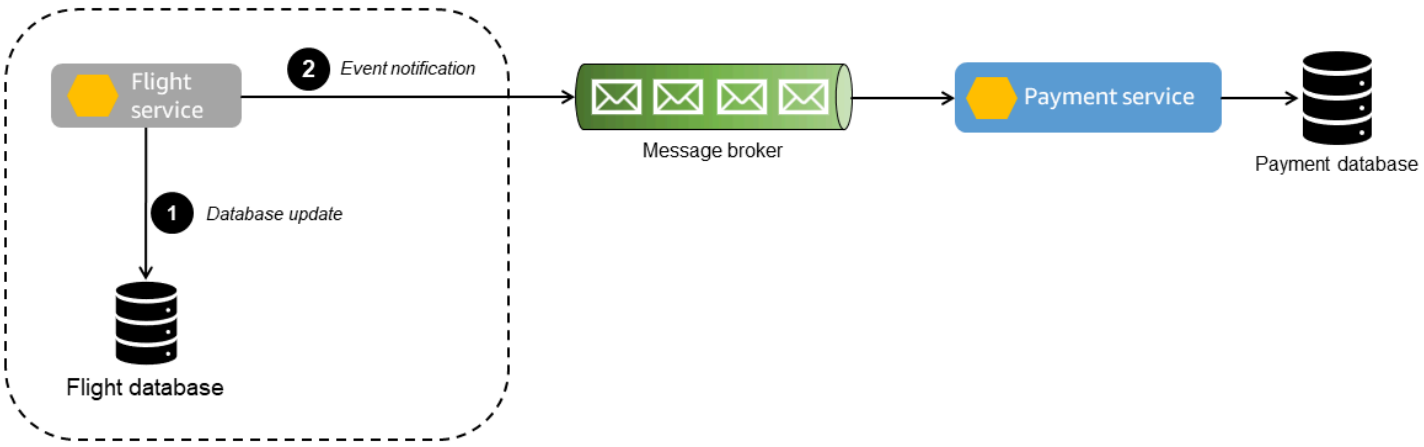
Problèmes et considérations

- Messages dupliqués : le service de traitement des événements peut envoyer des messages ou des événements en double. Nous vous recommandons donc de rendre le service consommateur idempotent en suivant les messages traités.
- Ordre de notification : envoyez des messages ou des événements dans l'ordre dans lequel le service met à jour la base de données. Cela est essentiel pour le modèle d'approvisionnement en événements, dans lequel vous pouvez utiliser un magasin d'événements pour point-in-time récupérer le magasin de données. Si la commande est incorrecte, cela peut compromettre la qualité des données. La cohérence à terme et la restauration de la base de données peuvent aggraver le problème si l'ordre des notifications n'est pas préservé.
- Restauration de transaction : n'envoyez pas de notification d'événement si la transaction est restaurée.
- Gestion des transactions au niveau du service : si la transaction couvre des services nécessitant des mises à jour des magasins de données, utilisez le modèle d'orchestration de saga pour préserver l'intégrité des données dans les magasins de données.

Mise en œuvre

Architecture de haut niveau

Le diagramme de séquence suivant montre l'ordre des événements qui se produisent lors des opérations d'écriture double.



1. Le service de vol écrit dans la base de données et envoie une notification d'événement au service de paiement.

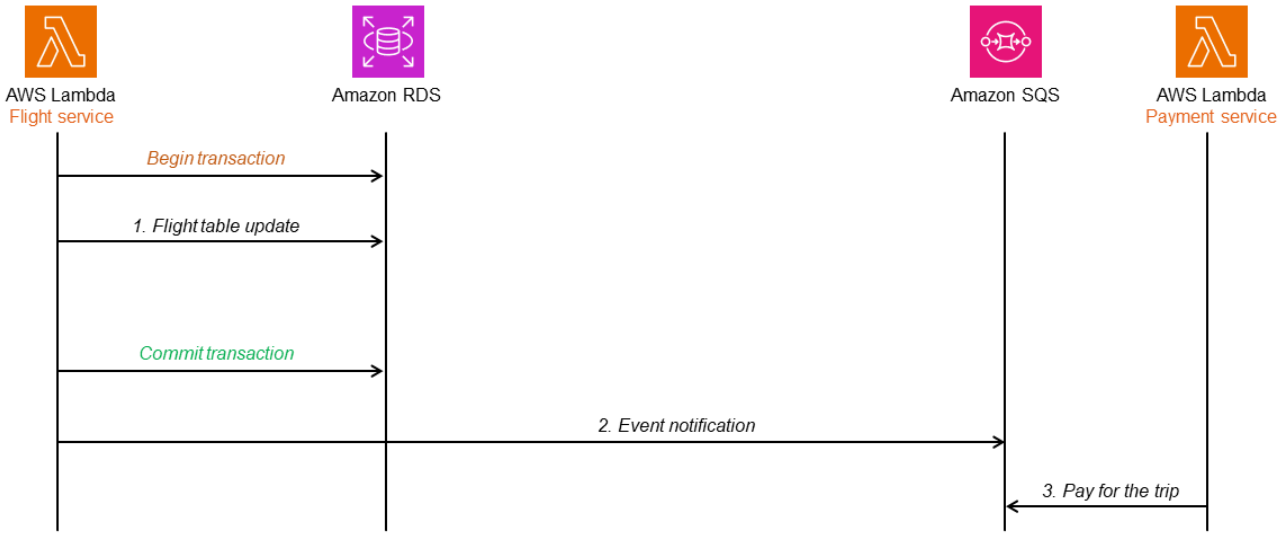
2. L'agent de messages transmet les messages et les événements au service de paiement. Toute défaillance de l'agent de messages empêche le service de paiement de recevoir les mises à jour.

Si la mise à jour de la base de données des vols échoue, mais que la notification est envoyée, le service de paiement traitera le paiement en fonction de la notification d'événement. Cela entraînera des incohérences dans les données en aval.

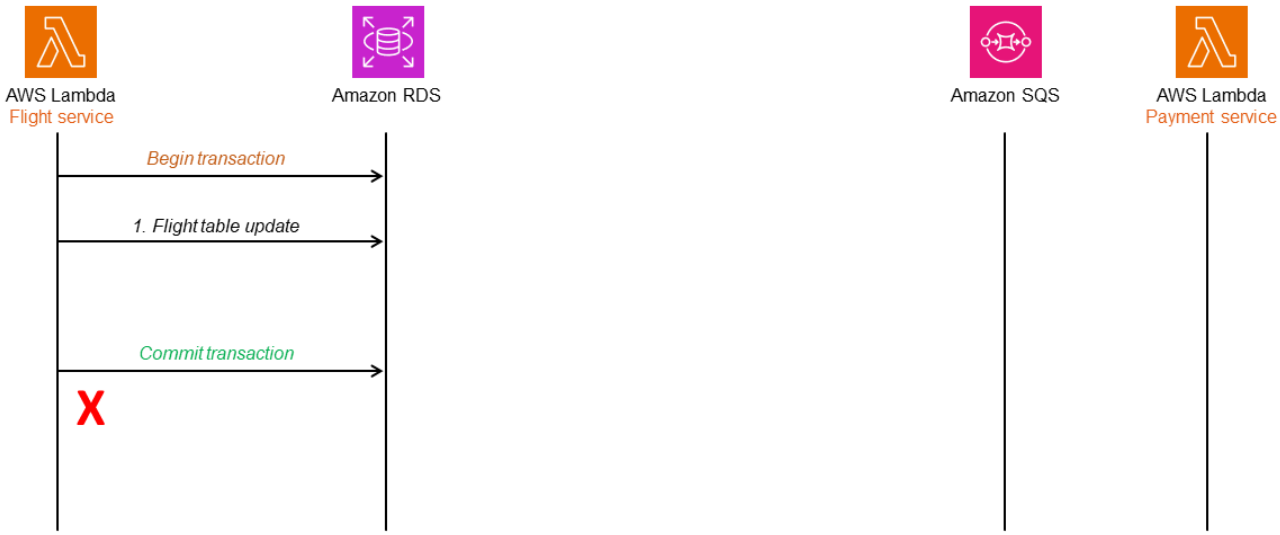
Mise en œuvre à l'aide des services AWS

Pour illustrer le schéma dans le diagramme de séquence, nous utiliserons les AWS services suivants, comme indiqué dans le schéma suivant.

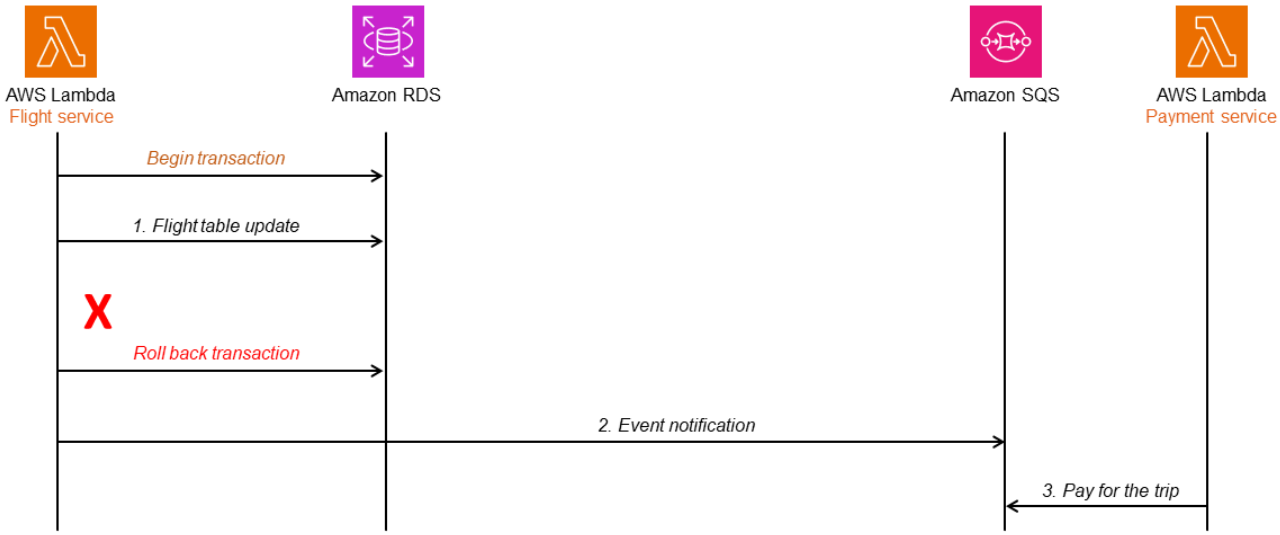
- Les microservices sont mis en œuvre à l'aide d'AWS Lambda.
- La base de données principale est gérée par Amazon Relational Database Service (Amazon RDS).
- Amazon Simple Queue Service (Amazon SQS) agit en tant qu'agent de messages qui reçoit les notifications d'événements.



Si le service de vol échoue après avoir validé la transaction, la notification d'événement peut ne pas être envoyée.



Cependant, la transaction peut échouer et être restaurée, mais la notification d'événement peut tout de même être envoyée, obligeant le service de paiement à traiter le paiement.



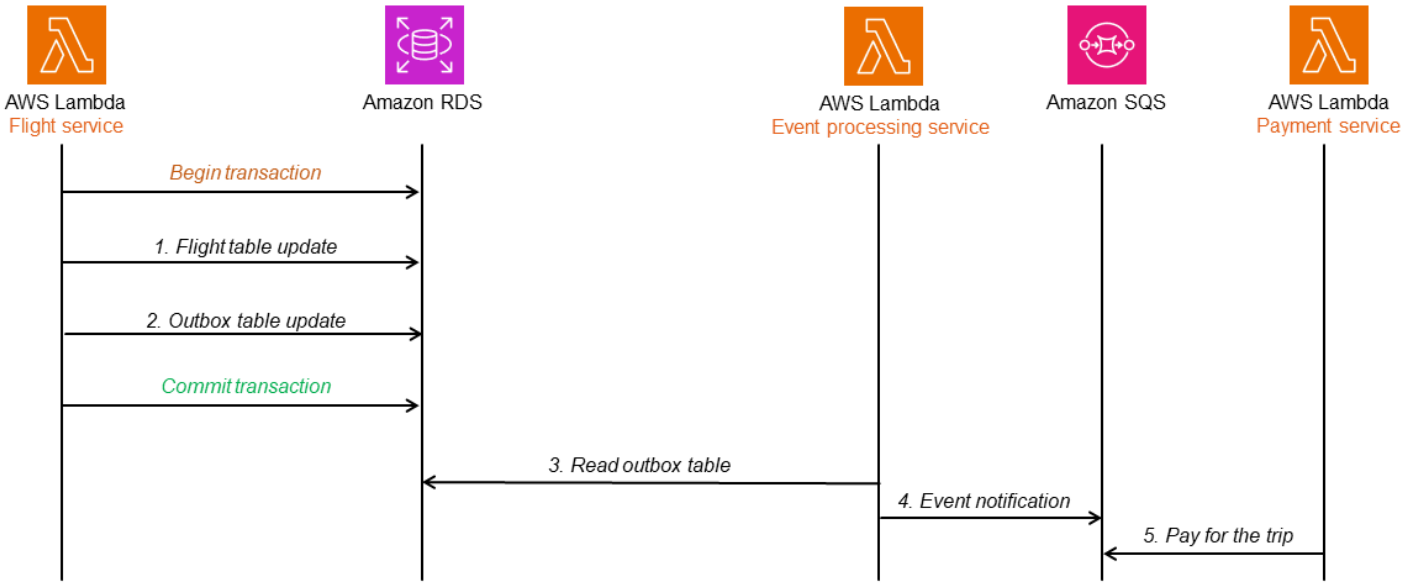
Pour résoudre ce problème, vous pouvez utiliser une table de boîte d'envoi ou la capture des données de modification (CDC). Les sections suivantes traitent de ces deux options et de la manière dont vous pouvez les mettre en œuvre à l'aide des services AWS.

Utilisation d'une table de boîte d'envoi avec une base de données relationnelle

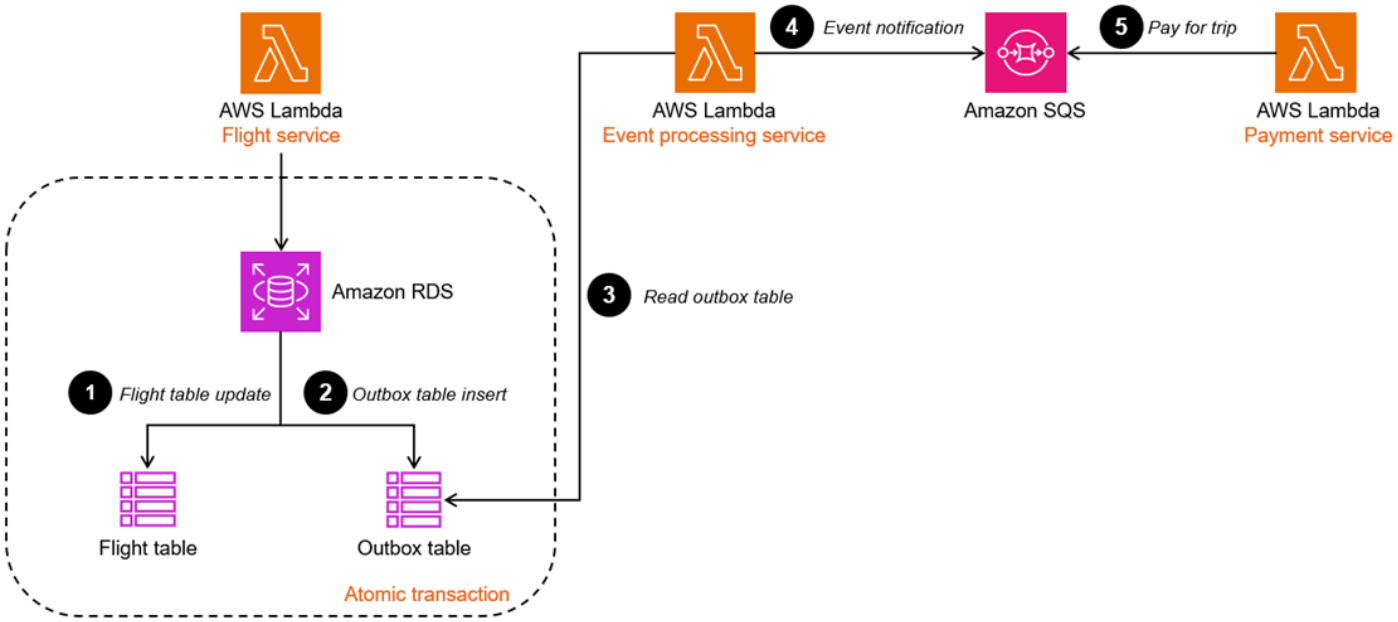
Une table de boîte d'envoi stocke tous les événements du service de vol avec un horodatage et un numéro de séquence.

Lorsque la table de vol est mise à jour, la table de boîte d'envoi est également mise à jour dans le cadre de la même transaction. Un autre service (par exemple, le service de traitement des événements) lit la table de la boîte d'envoi et envoie l'événement à Amazon SQS. Amazon SQS envoie un message concernant l'événement au service de paiement pour un traitement ultérieur. Les files d'attente standard Amazon SQS garantissent que le message est délivré au moins une fois et qu'il ne soit pas perdu. Toutefois, lorsque vous utilisez des files d'attente standard Amazon SQS, le même message ou événement peut être transmis plusieurs fois. Vous devez donc vous assurer que le service de notification d'événements est idempotent (c'est-à-dire que le traitement du même message plusieurs fois ne devrait pas avoir d'effet négatif). Si vous souhaitez que le message soit livré une seule fois, vous pouvez utiliser les files d'attente FIFO (premier entré, premier sorti) d'Amazon SQS dans le cadre de l'ordre des messages.

Si la mise à jour de la table de vol ou la mise à jour de la table de boîte d'envoi échouent, l'intégralité de la transaction est restaurée, de sorte qu'il n'y a aucune incohérence dans les données en aval.



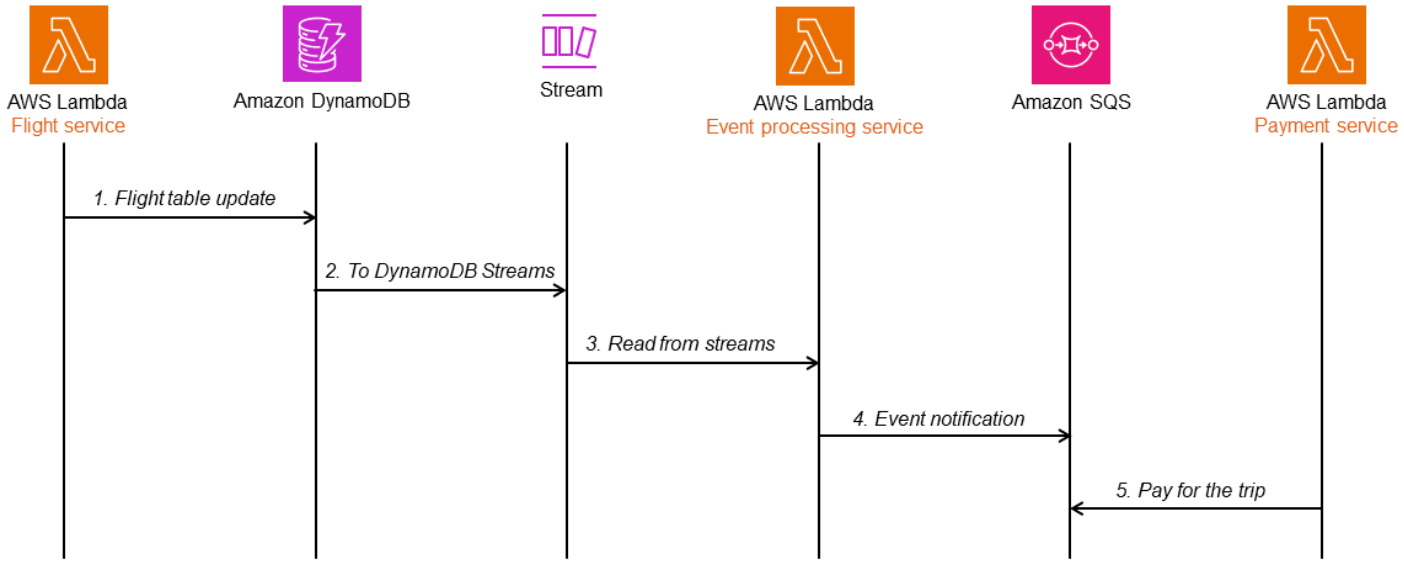
Dans le schéma suivant, l'architecture de boîte d'envoi transactionnelle est mise en œuvre à l'aide d'une base de données Amazon RDS. Lorsque le service de traitement des événements lit la table de la boîte d'envoi, il reconnaît uniquement les lignes faisant partie d'une transaction validée (réussie), puis place le message correspondant à l'événement dans la file d'attente SQS, qui est lue par le service de paiement pour un traitement ultérieur. Cette conception résout le problème des opérations d'écriture double et préserve l'ordre des messages et des événements en utilisant des horodatages et des numéros de séquence.



Utilisation de la capture des données de modification (CDC)

Certaines bases de données prennent en charge la publication de modifications au niveau des éléments afin de capturer les données modifiées. Vous pouvez identifier les éléments modifiés et envoyer une notification d'événement en conséquence. Cela permet d'économiser les frais liés à la création d'une autre table pour suivre les mises à jour. L'événement initié par le service de vol est enregistré dans un autre attribut du même élément.

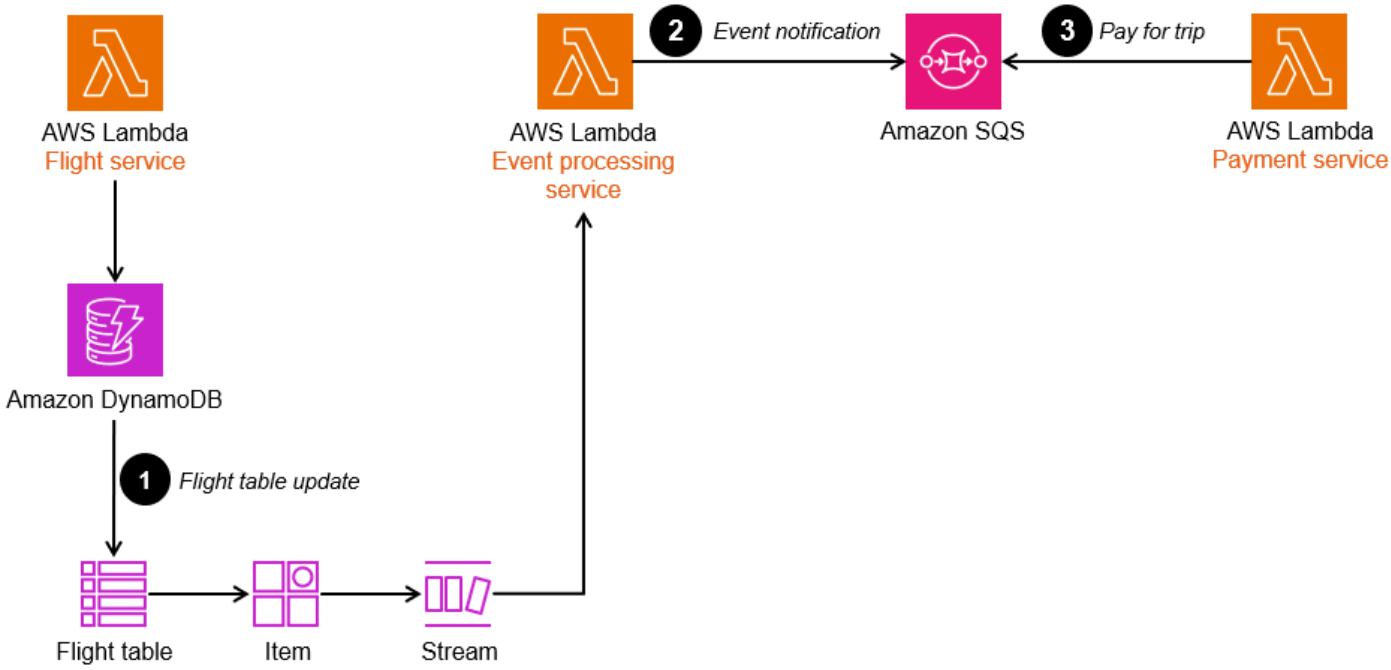
Amazon DynamoDB est une base de données NoSQL clé-valeur qui prend en charge les mises à jour CDC. Dans le diagramme de séquence suivant, DynamoDB publie les modifications apportées au niveau des éléments dans Amazon DynamoDB Streams. Le service de traitement des événements lit les flux et publie la notification d'événement dans le service de paiement pour un traitement ultérieur.



DynamoDB Streams capture le flux d'informations relatives aux modifications apportées au niveau des éléments dans une table DynamoDB à l'aide d'une séquence temporelle.

Vous pouvez implémenter un modèle de boîte d'envoi transactionnelle en activant les flux dans la table DynamoDB. La fonction Lambda du service de traitement des événements est associée à ces flux.

- Lorsque la table de vol est mise à jour, les données modifiées sont capturées par DynamoDB Streams, et le service de traitement des événements interroge le flux à la recherche de nouveaux enregistrements.
- Lorsque de nouveaux enregistrements de flux sont disponibles, la fonction Lambda place de manière synchrone le message de l'événement dans la file d'attente SQS pour un traitement ultérieur. Vous pouvez ajouter un attribut à l'élément DynamoDB pour capturer l'horodatage et le numéro de séquence selon les besoins afin d'améliorer la robustesse de l'implémentation.



Exemple de code

Utilisation d'une table de boîte d'envoi

L'exemple de code présenté dans cette section montre comment implémenter le modèle de boîte d'envoi transactionnelle à l'aide d'une table de boîte d'envoi. Pour consulter le code complet, consultez le GitHubréférentiel de cet exemple.

L'extrait de code suivant enregistre l'entité `Flight` et l'événement `Flight` dans la base de données dans leurs tables respectives au cours d'une seule transaction.

```
@PostMapping("/flights")
    @Transactional
    public Flight createFlight(@Valid @RequestBody Flight flight) {
        Flight savedFlight = flightRepository.save(flight);
        JsonNode flightPayload = objectMapper.convertValue(flight, JsonNode.class);
        FlightOutbox outboxEvent = new FlightOutbox(flight.getId().toString(),
 FlightOutbox.EventType.FLIGHT_BOOKED,
                flightPayload);
        outboxRepository.save(outboxEvent);
        return savedFlight;
    }
```

Un service distinct est chargé d'analyser régulièrement la table de la boîte d'envoi pour détecter de nouveaux événements, de les envoyer à Amazon SQS et de les supprimer de la table si Amazon SQS répond correctement. Le taux d'interrogation est configurable dans le fichier `application.properties`.

```
@Scheduled(fixedDelayString = "${sqs.polling_ms}")
    public void forwardEventsToSQS() {
        List<FlightOutbox> entities =
 outboxRepository.findAllByOrderByIdAsc(Pageable.ofSize(batchSize)).toList();
        if (!entities.isEmpty()) {
            GetQueueUrlRequest getQueueRequest = GetQueueUrlRequest.builder()
                    .queueName(sqsQueueName)
                    .build();
            String queueUrl = this.sqsClient.getQueueUrl(getQueueRequest).queueUrl();
            List<SendMessageBatchRequestEntry> messageEntries = new ArrayList<>();
            entities.forEach(entity ->
 messageEntries.add(SendMessageBatchRequestEntry.builder()
                    .id(entity.getId().toString())
                    .messageGroupId(entity.getAggregateId())
                    .messageDeduplicationId(entity.getId().toString())
                    .messageBody(entity.getPayload().toString())
                    .build())
            );
            SendMessageBatchRequest sendMessageBatchRequest =
 SendMessageBatchRequest.builder()
                    .queueUrl(queueUrl)
                    .entries(messageEntries)
                    .build();
            sqsClient.sendMessageBatch(sendMessageBatchRequest);
            outboxRepository.deleteAllInBatch(entities);
        }
    }
```

Utilisation de la capture des données de modification (CDC)

L'exemple de code présenté dans cette section montre comment implémenter le modèle de boîte d'envoi transactionnelle en utilisant les fonctionnalités de capture des données de modification (CDC) de DynamoDB. Pour consulter le code complet, consultez le GitHubréférentiel de cet exemple.

L'extrait de AWS Cloud Development Kit (AWS CDK) code suivant crée une table de vol DynamoDB et un flux de données Amazon Kinesis (`cdcStream`), et configure la table de vol pour envoyer toutes ses mises à jour au flux.

```
Const cdcStream = new kinesis.Stream(this, 'flightsCDCStream', {
    streamName: 'flightsCDCStream'
})

const flightTable = new dynamodb.Table(this, 'flight', {
    tableName: 'flight',
    kinesisStream: cdcStream,
    partitionKey: {
        name: 'id',
        type: dynamodb.AttributeType.STRING,
    }

});
```

L'extrait de code et la configuration suivants définissent une fonction Spring Cloud Stream qui récupère les mises à jour dans le flux Kinesis et transmet ces événements à une file d'attente SQS pour un traitement ultérieur.

```
applications.properties
spring.cloud.stream.bindings.sendToSQS-in-0.destination=${kinesisstreamname}
spring.cloud.stream.bindings.sendToSQS-in-0.content-type=application/ddb

QueueService.java
@Bean
public Consumer<Flight> sendToSQS() {
    return this::forwardEventsToSQS;
}

public void forwardEventsToSQS(Flight flight) {
    GetQueueUrlRequest getQueueRequest = GetQueueUrlRequest.builder()
            .queueName(sqsQueueName)
            .build();
    String queueUrl = this.sqsClient.getQueueUrl(getQueueRequest).queueUrl();
    try {
        SendMessageRequest send_msg_request = SendMessageRequest.builder()
                .queueUrl(queueUrl)
                .messageBody(objectMapper.writeValueAsString(flight))
                .messageGroupId("1")
                .messageDeduplicationId(flight.getId().toString())
                .build();
        sqsClient.sendMessage(send_msg_request);
    } catch (IOException | AmazonServiceException e) {
```

```
        logger.error("Error sending message to SQS", e);
    }
}
```

GitHub référentiel

Pour une implémentation complète de l'exemple d'architecture pour ce modèle, consultez le GitHub référentiel à l'adresse https://github.com/aws-samples/ transactional-outbox-pattern.

Ressources

Références

- AWSCentre d'architecture
- Centre pour développeurs AWS
- La bibliothèque Amazon Builders

Outils

- AWS Well-Architected Tool
- Conteneur AWS App2
- AWS Microservice Extractor pour .NET

Méthodologies

- L'application Twelve-Factor(ePub par Adam Wiggins)
- Nygard, Michaël T.Relâchez-le ! : Conception et déploiement de logiciels prêts pour la production. 2e éd. Raleigh, Caroline du Nord : Étagère pragmatique, 2018.
- Persistance polyglotte(billet de blog de Martin Fowler)
- StranglerFigApplication(billet de blog de Martin Fowler)

Historique du document

Le tableau suivant décrit les modifications importantes apportées à ce guide. Pour être averti des mises à jour à venir, abonnez-vous à un fil RSS.

Modification	Description	Date
Nouveaux modèles	Ajout de deux nouveaux modèles : architecture hexagonale et scatter-gather.	7 mai 2024
Nouveaux exemples de code	Ajout d'un exemple de code pour le cas d'utilisation de la capture des données de modification (CDC) au modèle de boîte d'envoi transacti onnelle.	23 février 2024
Nouveaux exemples de code	• Mise à jour du modèle de boîte d’envoi transacti onnelle avec un exemple de code. • Suppression de la section sur les modèles d’orchest ration et de chorégraphie, qui a été remplacée par la chorégraphie de saga et l’orchestration de saga.	16 novembre 2023
Nouveaux modèles	Trois nouveaux modèles ont été ajoutés : la chorégraphie de saga, publier/s’abonner et l’approvisionnement d’événement.	14 novembre 2023

Mettre à jour	Mise à jour de la section d'implémentation du modèle de figuier étrangleur.	2 octobre 2023
Publication initiale	Cette première version inclut huit modèles de conception : couche anticorruption (ACL), routage d'API, disjoncteur, orchestration et chorégraphie, réessai avec rétrogradation, orchestration de saga, fichier étrangleur et boîte d'envoi transactionnelle.	28 juillet 2023

AWS Glossaire des directives prescriptives

Les termes suivants sont couramment utilisés dans les stratégies, les guides et les modèles fournis par les directives AWS prescriptives. Pour suggérer des entrées, veuillez utiliser le lien Faire un commentaire à la fin du glossaire.

Nombres

7 R

Sept politiques de migration courantes pour transférer des applications vers le cloud. Ces politiques s'appuient sur les 5 R identifiés par Gartner en 2011 et sont les suivantes :

- Refactorisation/réarchitecture : transférez une application et modifiez son architecture en tirant pleinement parti des fonctionnalités natives cloud pour améliorer l'agilité, les performances et la capacité de mise à l'échelle. Cela implique généralement le transfert du système d'exploitation et de la base de données. Exemple : migrez votre base de données Oracle sur site vers l'édition compatible avec Amazon Aurora PostgreSQL.
- Replateformer (déplacer et remodeler) : transférez une application vers le cloud et introduisez un certain niveau d'optimisation pour tirer parti des fonctionnalités du cloud. Exemple : migrez votre base de données Oracle sur site vers Amazon Relational Database Service (Amazon RDS) pour Oracle dans le. AWS Cloud
- Racheter (rachat) : optez pour un autre produit, généralement en passant d'une licence traditionnelle à un modèle SaaS. Exemple : migrez votre système de gestion de la relation client (CRM) vers Salesforce.com.
- Réhéberger (lift and shift) : transférez une application vers le cloud sans apporter de modifications pour tirer parti des fonctionnalités du cloud. Exemple : migrez votre base de données Oracle sur site vers Oracle sur une instance EC2 dans le. AWS Cloud
- Relocaliser (lift and shift au niveau de l'hyperviseur) : transférez l'infrastructure vers le cloud sans acheter de nouveau matériel, réécrire des applications ou modifier vos opérations existantes. Vous migrez des serveurs d'une plateforme sur site vers un service cloud pour la même plateforme. Exemple : migrer une Microsoft Hyper-V application vers AWS.
- Retenir : conservez les applications dans votre environnement source. Il peut s'agir d'applications nécessitant une refactorisation majeure, que vous souhaitez retarder, et d'applications existantes que vous souhaitez retenir, car rien ne justifie leur migration sur le plan commercial.

- Retirer : mettez hors service ou supprimez les applications dont vous n'avez plus besoin dans votre environnement source.

A

ABAC

Voir contrôle d'accès basé sur les attributs.

services abstraits

Consultez la section Services gérés.

ACIDE

Voir atomicité, consistance, isolation, durabilité.

migration active-active

Méthode de migration de base de données dans laquelle la synchronisation des bases de données source et cible est maintenue (à l'aide d'un outil de réplication bidirectionnelle ou d'opérations d'écriture double), tandis que les deux bases de données gèrent les transactions provenant de la connexion d'applications pendant la migration. Cette méthode prend en charge la migration par petits lots contrôlés au lieu d'exiger un basculement ponctuel. Elle est plus flexible mais demande plus de travail qu'une migration active-passive.

migration active-passive

Méthode de migration de base de données dans laquelle la synchronisation des bases de données source et cible est maintenue, mais seule la base de données source gère les transactions provenant de la connexion d'applications pendant que les données sont répliquées vers la base de données cible. La base de données cible n'accepte aucune transaction pendant la migration.

fonction d'agrégation

Fonction SQL qui agit sur un groupe de lignes et calcule une valeur de retour unique pour le groupe. Des exemples de fonctions d'agrégation incluent `SUM` et`MAX`.

AI

Voir intelligence artificielle.

AIOps

Voir les opérations d'intelligence artificielle.

anonymisation

Processus de suppression définitive d'informations personnelles dans un ensemble de données. L'anonymisation peut contribuer à protéger la vie privée. Les données anonymisées ne sont plus considérées comme des données personnelles.

anti-motif

Solution fréquemment utilisée pour un problème récurrent lorsque la solution est contre-productive, inefficace ou moins efficace qu'une solution alternative.

contrôle des applications

Une approche de sécurité qui permet d'utiliser uniquement des applications approuvées afin de protéger un système contre les logiciels malveillants.

portefeuille d'applications

Ensemble d'informations détaillées sur chaque application utilisée par une organisation, y compris le coût de génération et de maintenance de l'application, ainsi que sa valeur métier. Ces informations sont essentielles pour le processus de découverte et d'analyse du portefeuille et permettent d'identifier et de prioriser les applications à migrer, à moderniser et à optimiser.

intelligence artificielle (IA)

Domaine de l'informatique consacré à l'utilisation des technologies de calcul pour exécuter des fonctions cognitives généralement associées aux humains, telles que l'apprentissage, la résolution de problèmes et la reconnaissance de modèles. Pour plus d'informations, veuillez consulter Qu'est-ce que l'intelligence artificielle ?

opérations d'intelligence artificielle (AIOps)

Processus consistant à utiliser des techniques de machine learning pour résoudre les problèmes opérationnels, réduire les incidents opérationnels et les interventions humaines, mais aussi améliorer la qualité du service. Pour plus d'informations sur la façon dont les AIOps sont utilisées dans la stratégie de migration AWS , veuillez consulter le guide d'intégration des opérations.

chiffrement asymétrique

Algorithme de chiffrement qui utilise une paire de clés, une clé publique pour le chiffrement et une clé privée pour le déchiffrement. Vous pouvez partager la clé publique, car elle n'est pas utilisée pour le déchiffrement, mais l'accès à la clé privée doit être très restreint.

atomicité, cohérence, isolement, durabilité (ACID)

Ensemble de propriétés logicielles garantissant la validité des données et la fiabilité opérationnelle d'une base de données, même en cas d'erreur, de panne de courant ou d'autres problèmes.

contrôle d'accès par attributs (ABAC)

Pratique qui consiste à créer des autorisations détaillées en fonction des attributs de l'utilisateur, tels que le service, le poste et le nom de l'équipe. Pour plus d'informations, consultez ABAC pour AWS dans la documentation AWS Identity and Access Management (IAM).

source de données faisant autorité

Emplacement où vous stockez la version principale des données, considérée comme la source d'information la plus fiable. Vous pouvez copier les données de la source de données officielle vers d'autres emplacements à des fins de traitement ou de modification des données, par exemple en les anonymisant, en les expurgant ou en les pseudonymisant.

Zone de disponibilité

Un emplacement distinct au sein d'un Région AWS réseau isolé des défaillances dans d'autres zones de disponibilité et fournissant une connectivité réseau peu coûteuse et à faible latence aux autres zones de disponibilité de la même région.

AWS Cadre d'adoption du cloud (AWS CAF)

Un cadre de directives et de meilleures pratiques visant AWS à aider les entreprises à élaborer un plan efficace pour réussir leur migration vers le cloud. AWS La CAF organise ses conseils en six domaines prioritaires appelés perspectives : les affaires, les personnes, la gouvernance, les plateformes, la sécurité et les opérations. Les perspectives d'entreprise, de personnes et de gouvernance mettent l'accent sur les compétences et les processus métier, tandis que les perspectives relatives à la plateforme, à la sécurité et aux opérations se concentrent sur les compétences et les processus techniques. Par exemple, la perspective liée aux personnes cible les parties prenantes qui s'occupent des ressources humaines (RH), des fonctions de dotation en personnel et de la gestion des personnes. Dans cette perspective, la AWS CAF fournit des conseils pour le développement du personnel, la formation et les communications afin de préparer l'organisation à une adoption réussie du cloud. Pour plus d'informations, veuillez consulter le site Web AWS CAF et le livre blanc AWS CAF.

AWS Cadre de qualification de la charge de travail (AWS WQF)

Outil qui évalue les charges de travail liées à la migration des bases de données, recommande des stratégies de migration et fournit des estimations de travail. AWS Le WQF est inclus avec

AWS Schema Conversion Tool (AWS SCT). Il analyse les schémas de base de données et les objets de code, le code d'application, les dépendances et les caractéristiques de performance, et fournit des rapports d'évaluation.

B

mauvais bot

Un bot destiné à perturber ou à nuire à des individus ou à des organisations.

BCP

Consultez la section Planification de la continuité des activités.

graphique de comportement

Vue unifiée et interactive des comportements des ressources et des interactions au fil du temps. Vous pouvez utiliser un graphique de comportement avec Amazon Detective pour examiner les tentatives de connexion infructueuses, les appels d'API suspects et les actions similaires. Pour plus d'informations, veuillez consulter Data in a behavior graph dans la documentation Detective.

système de poids fort

Système qui stocke d'abord l'octet le plus significatif. Voir aussi endianité.

classification binaire

Processus qui prédit un résultat binaire (l'une des deux classes possibles). Par exemple, votre modèle de machine learning peut avoir besoin de prévoir des problèmes tels que « Cet e-mail est-il du spam ou non ? » ou « Ce produit est-il un livre ou une voiture ? ».

filtre de Bloom

Structure de données probabiliste et efficace en termes de mémoire qui est utilisée pour tester si un élément fait partie d'un ensemble.

déploiement bleu/vert

Stratégie de déploiement dans laquelle vous créez deux environnements distincts mais identiques. Vous exécutez la version actuelle de l'application dans un environnement (bleu) et la nouvelle version de l'application dans l'autre environnement (vert). Cette stratégie vous permet de revenir rapidement en arrière avec un impact minimal.

bot

Application logicielle qui exécute des tâches automatisées sur Internet et simule l'activité ou l'interaction humaine. Certains robots sont utiles ou bénéfiques, comme les robots d'exploration Web qui indexent des informations sur Internet. D'autres robots, connus sous le nom de mauvais robots, sont destinés à perturber ou à nuire à des individus ou à des organisations.

botnet

Réseaux de robots infectés par des logiciels malveillants et contrôlés par une seule entité, connue sous le nom d'herder ou d'opérateur de bots. Les botnets sont le mécanisme le plus connu pour faire évoluer les bots et leur impact.

branche

Zone contenue d'un référentiel de code. La première branche créée dans un référentiel est la branche principale. Vous pouvez créer une branche à partir d'une branche existante, puis développer des fonctionnalités ou corriger des bogues dans la nouvelle branche. Une branche que vous créez pour générer une fonctionnalité est communément appelée branche de fonctionnalités. Lorsque la fonctionnalité est prête à être publiée, vous fusionnez à nouveau la branche de fonctionnalités dans la branche principale. Pour plus d'informations, consultez À propos des branches (GitHub documentation).

accès par brise-vitre

Dans des circonstances exceptionnelles et par le biais d'un processus approuvé, c'est un moyen rapide pour un utilisateur d'accéder à un accès auquel Compte AWS il n'est généralement pas autorisé. Pour plus d'informations, consultez l'indicateur Implementation break-glass procedures dans le guide Well-Architected AWS .

stratégie existante (brownfield)

L'infrastructure existante de votre environnement. Lorsque vous adoptez une stratégie existante pour une architecture système, vous concevez l'architecture en fonction des contraintes des systèmes et de l'infrastructure actuels. Si vous étendez l'infrastructure existante, vous pouvez combiner des politiques brownfield (existantes) et greenfield (inédites).

cache de tampon

Zone de mémoire dans laquelle sont stockées les données les plus fréquemment consultées.

capacité métier

Ce que fait une entreprise pour générer de la valeur (par exemple, les ventes, le service client ou le marketing). Les architectures de microservices et les décisions de développement

peuvent être dictées par les capacités métier. Pour plus d'informations, veuillez consulter la section Organisation en fonction des capacités métier du livre blanc Exécution de microservices conteneurisés sur AWS.

planification de la continuité des activités (BCP)

Plan qui tient compte de l'impact potentiel d'un événement perturbateur, tel qu'une migration à grande échelle, sur les opérations, et qui permet à une entreprise de reprendre ses activités rapidement.

C

CAF

Voir le cadre d'adoption du AWS cloud.

déploiement de Canary

Diffusion lente et progressive d'une version pour les utilisateurs finaux. Lorsque vous êtes sûr, vous déployez la nouvelle version et remplacez la version actuelle dans son intégralité.

CCoE

Voir le Centre d'excellence du cloud.

CDC

Consultez la section Capture des données de modification.

capture des données de modification (CDC)

Processus de suivi des modifications apportées à une source de données, telle qu'une table de base de données, et d'enregistrement des métadonnées relatives à ces modifications. Vous pouvez utiliser la CDC à diverses fins, telles que l'audit ou la réplication des modifications dans un système cible afin de maintenir la synchronisation.

ingénierie du chaos

Introduire intentionnellement des défaillances ou des événements perturbateurs pour tester la résilience d'un système. Vous pouvez utiliser AWS Fault Injection Service (AWS FIS) pour effectuer des expériences qui stressent vos AWS charges de travail et évaluer leur réponse.

CI/CD

Découvrez l'intégration continue et la livraison continue.

classification

Processus de catégorisation qui permet de générer des prédictions. Les modèles de ML pour les problèmes de classification prédisent une valeur discrète. Les valeurs discrètes se distinguent toujours les unes des autres. Par exemple, un modèle peut avoir besoin d'évaluer la présence ou non d'une voiture sur une image.

chiffrement côté client

Chiffrement des données localement, avant que la cible ne les Service AWS reçoive.

Centre d'excellence cloud (CCoE)

Une équipe multidisciplinaire qui dirige les efforts d'adoption du cloud au sein d'une organisation, notamment en développant les bonnes pratiques en matière de cloud, en mobilisant des ressources, en établissant des délais de migration et en guidant l'organisation dans le cadre de transformations à grande échelle. Pour plus d'informations, consultez les articles du CCoE sur le blog de stratégie AWS Cloud d'entreprise.

cloud computing

Technologie cloud généralement utilisée pour le stockage de données à distance et la gestion des appareils IoT. Le cloud computing est généralement associé à la technologie informatique de pointe.

modèle d'exploitation du cloud

Dans une organisation informatique, modèle d'exploitation utilisé pour créer, faire évoluer et optimiser un ou plusieurs environnements cloud. Pour plus d'informations, consultez la section Création de votre modèle d'exploitation cloud.

étapes d'adoption du cloud

Les quatre phases que les entreprises traversent généralement lorsqu'elles migrent vers AWS Cloud :

- Projet : exécution de quelques projets liés au cloud à des fins de preuve de concept et d'apprentissage
- Base : réaliser des investissements fondamentaux pour mettre à l'échelle l'adoption du cloud (par exemple, en créant une zone de destination, en définissant un CCoE ou en établissant un modèle opérationnel)
- Migration : migration d'applications individuelles

- Réinvention : optimisation des produits et services et innovation dans le cloud

Ces étapes ont été définies par Stephen Orban dans le billet de blog The Journey Toward Cloud-First & the Stages of Adoption publié sur le blog AWS Cloud Enterprise Strategy. Pour plus d'informations sur leur lien avec la stratégie de AWS migration, consultez le guide de préparation à la migration.

CMDB

Voir base de données de gestion de configuration.

référentiel de code

Emplacement où le code source et d'autres ressources, comme la documentation, les exemples et les scripts, sont stockés et mis à jour par le biais de processus de contrôle de version. Les référentiels cloud courants incluent GitHub ou AWS CodeCommit. Chaque version du code est appelée branche. Dans une structure de microservice, chaque référentiel est consacré à une seule fonctionnalité. Un seul pipeline CI/CD peut utiliser plusieurs référentiels.

cache passif

Cache tampon vide, mal rempli ou contenant des données obsolètes ou non pertinentes. Cela affecte les performances, car l'instance de base de données doit lire à partir de la mémoire principale ou du disque, ce qui est plus lent que la lecture à partir du cache tampon.

données gelées

Données rarement consultées et généralement historiques. Lorsque vous interrogez ce type de données, les requêtes lentes sont généralement acceptables. Le transfert de ces données vers des niveaux ou classes de stockage moins performants et moins coûteux peut réduire les coûts.

vision par ordinateur (CV)

Domaine de l'IA qui utilise l'apprentissage automatique pour analyser et extraire des informations à partir de formats visuels tels que des images numériques et des vidéos. Par exemple, AWS Panorama propose des appareils qui ajoutent des CV aux réseaux de caméras locaux, et Amazon SageMaker fournit des algorithmes de traitement d'image pour les CV.

dérive de configuration

Pour une charge de travail, une modification de configuration par rapport à l'état attendu. Cela peut entraîner une non-conformité de la charge de travail, et cela est généralement progressif et involontaire.

base de données de gestion des configurations (CMDB)

Référentiel qui stocke et gère les informations relatives à une base de données et à son environnement informatique, y compris les composants matériels et logiciels ainsi que leurs configurations. Vous utilisez généralement les données d'une CMDB lors de la phase de découverte et d'analyse du portefeuille de la migration.

pack de conformité

Ensemble de AWS Config règles et d'actions correctives que vous pouvez assembler pour personnaliser vos contrôles de conformité et de sécurité. Vous pouvez déployer un pack de conformité en tant qu'entité unique dans une région Compte AWS et, ou au sein d'une organisation, à l'aide d'un modèle YAML. Pour plus d'informations, consultez la section Packs de conformité dans la AWS Config documentation.

intégration continue et livraison continue (CI/CD)

Processus d'automatisation des étapes source, de génération, de test, intermédiaire et de production du processus de publication du logiciel. CI/CD est communément décrit comme un pipeline. CI/CD peut vous aider à automatiser les processus, à améliorer la productivité, à améliorer la qualité du code et à accélérer les livraisons. Pour plus d'informations, veuillez consulter Avantages de la livraison continue. CD peut également signifier déploiement continu. Pour plus d'informations, veuillez consulter Livraison continue et déploiement continu.

CV

Voir vision par ordinateur.

D

données au repos

Données stationnaires dans votre réseau, telles que les données stockées.

classification des données

Processus permettant d'identifier et de catégoriser les données de votre réseau en fonction de leur sévérité et de leur sensibilité. Il s'agit d'un élément essentiel de toute stratégie de gestion des risques de cybersécurité, car il vous aide à déterminer les contrôles de protection et de conservation appropriés pour les données. La classification des données est une composante du pilier de sécurité du AWS Well-Architected Framework. Pour plus d'informations, veuillez consulter Classification des données.

dérive des données

Une variation significative entre les données de production et les données utilisées pour entraîner un modèle ML, ou une modification significative des données d'entrée au fil du temps. La dérive des données peut réduire la qualité, la précision et l'équité globales des prédictions des modèles ML.

données en transit

Données qui circulent activement sur votre réseau, par exemple entre les ressources du réseau.

maillage de données

Un cadre architectural qui fournit une propriété des données distribuée et décentralisée avec une gestion et une gouvernance centralisées.

minimisation des données

Le principe de collecte et de traitement des seules données strictement nécessaires. La pratique de la minimisation des données AWS Cloud peut réduire les risques liés à la confidentialité, les coûts et l'empreinte carbone de vos analyses.

périmètre de données

Ensemble de garde-fous préventifs dans votre AWS environnement qui permettent de garantir que seules les identités fiables accèdent aux ressources fiables des réseaux attendus. Pour plus d'informations, voir Création d'un périmètre de données sur AWS.

prétraitement des données

Pour transformer les données brutes en un format facile à analyser par votre modèle de ML. Le prétraitement des données peut impliquer la suppression de certaines colonnes ou lignes et le traitement des valeurs manquantes, incohérentes ou en double.

provenance des données

Le processus de suivi de l'origine et de l'historique des données tout au long de leur cycle de vie, par exemple la manière dont les données ont été générées, transmises et stockées.

sujet des données

Personne dont les données sont collectées et traitées.

entrepôt des données

Un système de gestion des données qui prend en charge les informations commerciales, telles que les analyses. Les entrepôts de données contiennent généralement de grandes quantités de données historiques et sont généralement utilisés pour les requêtes et les analyses.

langage de définition de base de données (DDL)

Instructions ou commandes permettant de créer ou de modifier la structure des tables et des objets dans une base de données.

langage de manipulation de base de données (DML)

Instructions ou commandes permettant de modifier (insérer, mettre à jour et supprimer) des informations dans une base de données.

DDL

Voir langage de définition de base de données.

ensemble profond

Sert à combiner plusieurs modèles de deep learning à des fins de prédiction. Vous pouvez utiliser des ensembles profonds pour obtenir une prévision plus précise ou pour estimer l'incertitude des prédictions.

deep learning

Un sous-champ de ML qui utilise plusieurs couches de réseaux neuronaux artificiels pour identifier le mappage entre les données d'entrée et les variables cibles d'intérêt.

defense-in-depth

Approche de la sécurité de l'information dans laquelle une série de mécanismes et de contrôles de sécurité sont judicieusement répartis sur l'ensemble d'un réseau informatique afin de protéger la confidentialité, l'intégrité et la disponibilité du réseau et des données qu'il contient. Lorsque vous adoptez cette stratégie AWS, vous ajoutez plusieurs contrôles à différentes couches de la AWS Organizations structure afin de sécuriser les ressources. Par exemple, une defense-in-depth approche peut combiner l'authentification multifactorielle, la segmentation du réseau et le chiffrement.

administrateur délégué

Dans AWS Organizations, un service compatible peut enregistrer un compte AWS membre pour administrer les comptes de l'organisation et gérer les autorisations pour ce service. Ce compte est

appelé administrateur délégué pour ce service. Pour plus d'informations et une liste des services compatibles, veuillez consulter la rubrique Services qui fonctionnent avec AWS Organizations dans la documentation AWS Organizations .

déploiement

Processus de mise à disposition d'une application, de nouvelles fonctionnalités ou de corrections de code dans l'environnement cible. Le déploiement implique la mise en œuvre de modifications dans une base de code, puis la génération et l'exécution de cette base de code dans les environnements de l'application.

environnement de développement

Voir environnement.

contrôle de détection

Contrôle de sécurité conçu pour détecter, journaliser et alerter après la survenue d'un événement. Ces contrôles constituent une deuxième ligne de défense et vous alertent en cas d'événements de sécurité qui ont contourné les contrôles préventifs en place. Pour plus d'informations, veuillez consulter la rubrique Contrôles de détection dans Implementing security controls on AWS.

cartographie de la chaîne de valeur du développement (DVSM)

Processus utilisé pour identifier et hiérarchiser les contraintes qui nuisent à la rapidité et à la qualité du cycle de vie du développement logiciel. DVSM étend le processus de cartographie de la chaîne de valeur initialement conçu pour les pratiques de production allégée. Il met l'accent sur les étapes et les équipes nécessaires pour créer et transférer de la valeur tout au long du processus de développement logiciel.

jumeau numérique

Représentation virtuelle d'un système réel, tel qu'un bâtiment, une usine, un équipement industriel ou une ligne de production. Les jumeaux numériques prennent en charge la maintenance prédictive, la surveillance à distance et l'optimisation de la production.

tableau des dimensions

Dans un schéma en étoile, table plus petite contenant les attributs de données relatifs aux données quantitatives d'une table de faits. Les attributs des tables de dimensions sont généralement des champs de texte ou des nombres discrets qui se comportent comme du texte. Ces attributs sont couramment utilisés pour la contrainte des requêtes, le filtrage et l'étiquetage des ensembles de résultats.

catastrophe

Un événement qui empêche une charge de travail ou un système d'atteindre ses objectifs commerciaux sur son site de déploiement principal. Ces événements peuvent être des catastrophes naturelles, des défaillances techniques ou le résultat d'actions humaines, telles qu'une mauvaise configuration involontaire ou une attaque de logiciel malveillant.

reprise après sinistre (DR)

La stratégie et le processus que vous utilisez pour minimiser les temps d'arrêt et les pertes de données causés par un sinistre. Pour plus d'informations, consultez Disaster Recovery of Workloads on AWS : Recovery in the Cloud in the AWS Well-Architected Framework.

DML

Voir langage de manipulation de base de données.

conception axée sur le domaine

Approche visant à développer un système logiciel complexe en connectant ses composants à des domaines évolutifs, ou objectifs métier essentiels, que sert chaque composant. Ce concept a été introduit par Eric Evans dans son ouvrage Domain-Driven Design: Tackling Complexity in the Heart of Software (Boston : Addison-Wesley Professional, 2003). Pour plus d'informations sur l'utilisation du design piloté par domaine avec le modèle de figuier étrangleur, veuillez consulter Modernizing legacy Microsoft ASP.NET (ASMX) web services incrementally by using containers and Amazon API Gateway.

DR

Consultez la section Reprise après sinistre.

détection de dérive

Suivi des écarts par rapport à une configuration de référence. Par exemple, vous pouvez l'utiliser AWS CloudFormation pour détecter la dérive des ressources du système ou AWS Control Tower pour détecter les modifications de votre zone d'atterrissage susceptibles d'affecter le respect des exigences de gouvernance.

DVSM

Voir la cartographie de la chaîne de valeur du développement.

E

EDA

Voir analyse exploratoire des données.

informatique de périphérie

Technologie qui augmente la puissance de calcul des appareils intelligents en périphérie d'un réseau IoT. Comparé au cloud computing, l'informatique de pointe peut réduire la latence des communications et améliorer le temps de réponse.

chiffrement

Processus informatique qui transforme des données en texte clair, lisibles par l'homme, en texte chiffré.

clé de chiffrement

Chaîne cryptographique de bits aléatoires générée par un algorithme cryptographique. La longueur des clés peut varier, et chaque clé est conçue pour être imprévisible et unique.

endianisme

Ordre selon lequel les octets sont stockés dans la mémoire de l'ordinateur. Les systèmes de poids fort stockent d'abord l'octet le plus significatif. Les systèmes de poids faible stockent d'abord l'octet le moins significatif.

point de terminaison

Voir point de terminaison de service.

service de point de terminaison

Service que vous pouvez héberger sur un cloud privé virtuel (VPC) pour le partager avec d'autres utilisateurs. Vous pouvez créer un service de point de terminaison avec AWS PrivateLink et accorder des autorisations à d'autres principaux Comptes AWS ou à AWS Identity and Access Management (IAM) principaux. Ces comptes ou principaux peuvent se connecter à votre service de point de terminaison de manière privée en créant des points de terminaison d'un VPC d'interface. Pour plus d'informations, veuillez consulter Création d'un service de point de terminaison dans la documentation Amazon Virtual Private Cloud (Amazon VPC).

planification des ressources d'entreprise (ERP)

Système qui automatise et gère les principaux processus métier (tels que la comptabilité, le MES et la gestion de projet) pour une entreprise.

chiffrement d'enveloppe

Processus de chiffrement d'une clé de chiffrement à l'aide d'une autre clé de chiffrement. Pour plus d'informations, consultez la section Chiffrement des enveloppes dans la documentation AWS Key Management Service (AWS KMS).

environment

Instance d'une application en cours d'exécution. Les types d'environnement les plus courants dans le cloud computing sont les suivants :

- Environnement de développement : instance d'une application en cours d'exécution à laquelle seule l'équipe principale chargée de la maintenance de l'application peut accéder. Les environnements de développement sont utilisés pour tester les modifications avant de les promouvoir dans les environnements supérieurs. Ce type d'environnement est parfois appelé environnement de test.
- Environnements inférieurs : tous les environnements de développement d'une application, tels que ceux utilisés pour les générations et les tests initiaux.
- Environnement de production : instance d'une application en cours d'exécution à laquelle les utilisateurs finaux peuvent accéder. Dans un pipeline CI/CD, l'environnement de production est le dernier environnement de déploiement.
- Environnements supérieurs : tous les environnements accessibles aux utilisateurs autres que l'équipe de développement principale. Ils peuvent inclure un environnement de production, des environnements de préproduction et des environnements pour les tests d'acceptation par les utilisateurs.

épopée

Dans les méthodologies agiles, catégories fonctionnelles qui aident à organiser et à prioriser votre travail. Les épopées fournissent une description détaillée des exigences et des tâches d'implémentation. Par exemple, les points forts de la AWS CAF en matière de sécurité incluent la gestion des identités et des accès, les contrôles de détection, la sécurité des infrastructures, la protection des données et la réponse aux incidents. Pour plus d'informations sur les épopées dans la stratégie de migration AWS , veuillez consulter le guide d'implémentation du programme.

ERP

Voir Planification des ressources d'entreprise.

analyse exploratoire des données (EDA)

Processus d'analyse d'un jeu de données pour comprendre ses principales caractéristiques. Vous collectez ou agrégez des données, puis vous effectuez des enquêtes initiales pour trouver des modèles, détecter des anomalies et vérifier les hypothèses. L'EDA est réalisée en calculant des statistiques récapitulatives et en créant des visualisations de données.

F

tableau des faits

La table centrale dans un schéma en étoile. Il stocke des données quantitatives sur les opérations commerciales. Généralement, une table de faits contient deux types de colonnes : celles qui contiennent des mesures et celles qui contiennent une clé étrangère pour une table de dimensions.

échouer rapidement

Une philosophie qui utilise des tests fréquents et progressifs pour réduire le cycle de vie du développement. C'est un élément essentiel d'une approche agile.

limite d'isolation des défauts

Dans le AWS Cloud, une limite telle qu'une zone de disponibilité Région AWS, un plan de contrôle ou un plan de données qui limite l'effet d'une panne et contribue à améliorer la résilience des charges de travail. Pour plus d'informations, consultez la section Limites d'isolation des AWS pannes.

branche de fonctionnalités

Voir la succursale.

fonctionnalités

Les données d'entrée que vous utilisez pour faire une prédiction. Par exemple, dans un contexte de fabrication, les fonctionnalités peuvent être des images capturées périodiquement à partir de la ligne de fabrication.

importance des fonctionnalités

Le niveau d'importance d'une fonctionnalité pour les prédictions d'un modèle. Il s'exprime généralement sous la forme d'un score numérique qui peut être calculé à l'aide de différentes

techniques, telles que la méthode Shapley Additive Explanations (SHAP) et les gradients intégrés. Pour plus d'informations, voir Interprétabilité du modèle d'apprentissage automatique avec :AWS.

transformation de fonctionnalité

Optimiser les données pour le processus de ML, notamment en enrichissant les données avec des sources supplémentaires, en mettant à l'échelle les valeurs ou en extrayant plusieurs ensembles d'informations à partir d'un seul champ de données. Cela permet au modèle de ML de tirer parti des données. Par exemple, si vous décomposez la date « 2021-05-27 00:15:37 » en « 2021 », « mai », « jeudi » et « 15 », vous pouvez aider l'algorithme d'apprentissage à apprendre des modèles nuancés associés à différents composants de données.

FGAC

Découvrez le contrôle d'accès détaillé.

contrôle d'accès détaillé (FGAC)

Utilisation de plusieurs conditions pour autoriser ou refuser une demande d'accès.

migration instantanée (flash-cut)

Méthode de migration de base de données qui utilise la réplication continue des données via la capture des données de modification afin de migrer les données dans les plus brefs délais, au lieu d'utiliser une approche progressive. L'objectif est de réduire au maximum les temps d'arrêt.

G

blocage géographique

Voir les restrictions géographiques.

restrictions géographiques (blocage géographique)

Sur Amazon CloudFront, option permettant d'empêcher les utilisateurs de certains pays d'accéder aux distributions de contenu. Vous pouvez utiliser une liste d'autorisation ou une liste de blocage pour spécifier les pays approuvés et interdits. Pour plus d'informations, consultez la section Restreindre la distribution géographique de votre contenu dans la CloudFront documentation.

Flux de travail Gitflow

Approche dans laquelle les environnements inférieurs et supérieurs utilisent différentes branches dans un référentiel de code source. Le flux de travail Gitflow est considéré comme existant, et le flux de travail basé sur les troncs est l'approche moderne préférée.

stratégie inédite

L'absence d'infrastructures existantes dans un nouvel environnement. Lorsque vous adoptez une stratégie inédite pour une architecture système, vous pouvez sélectionner toutes les nouvelles technologies sans restriction de compatibilité avec l'infrastructure existante, également appelée brownfield. Si vous étendez l'infrastructure existante, vous pouvez combiner des politiques brownfield (existantes) et greenfield (inédites).

barrière de protection

Règle de haut niveau qui permet de régir les ressources, les politiques et la conformité au sein des unités d'organisation (UO). Les barrières de protection préventives appliquent des politiques pour garantir l'alignement sur les normes de conformité. Elles sont mises en œuvre à l'aide de politiques de contrôle des services et de limites des autorisations IAM. Les barrières de protection de détection détectent les violations des politiques et les problèmes de conformité, et génèrent des alertes pour y remédier. Ils sont implémentés à l'aide d'Amazon AWS Config AWS Security Hub GuardDuty AWS Trusted Advisor, d'Amazon Inspector et de AWS Lambda contrôles personnalisés.

H

HA

Découvrez la haute disponibilité.

migration de base de données hétérogène

Migration de votre base de données source vers une base de données cible qui utilise un moteur de base de données différent (par exemple, Oracle vers Amazon Aurora). La migration hétérogène fait généralement partie d'un effort de réarchitecture, et la conversion du schéma peut s'avérer une tâche complexe. AWS propose AWS SCT qui facilite les conversions de schémas.

haute disponibilité (HA)

Capacité d'une charge de travail à fonctionner en continu, sans intervention, en cas de difficultés ou de catastrophes. Les systèmes HA sont conçus pour basculer automatiquement, fournir constamment des performances de haute qualité et gérer différentes charges et défaillances avec un impact minimal sur les performances.

modernisation de l'historien

Approche utilisée pour moderniser et mettre à niveau les systèmes de technologie opérationnelle (OT) afin de mieux répondre aux besoins de l'industrie manufacturière. Un historien est un type de base de données utilisé pour collecter et stocker des données provenant de diverses sources dans une usine.

migration de base de données homogène

Migration de votre base de données source vers une base de données cible qui partage le même moteur de base de données (par exemple, Microsoft SQL Server vers Amazon RDS for SQL Server). La migration homogène s'inscrit généralement dans le cadre d'un effort de réhébergement ou de replateforme. Vous pouvez utiliser les utilitaires de base de données natifs pour migrer le schéma.

données chaudes

Données fréquemment consultées, telles que les données en temps réel ou les données translationnelles récentes. Ces données nécessitent généralement un niveau ou une classe de stockage à hautes performances pour fournir des réponses rapides aux requêtes.

correctif

Solution d'urgence à un problème critique dans un environnement de production. En raison de son urgence, un correctif est généralement créé en dehors du flux de travail de DevOps publication habituel.

période de soins intensifs

Immédiatement après le basculement, période pendant laquelle une équipe de migration gère et surveille les applications migrées dans le cloud afin de résoudre les problèmes éventuels. En règle générale, cette période dure de 1 à 4 jours. À la fin de la période de soins intensifs, l'équipe de migration transfère généralement la responsabilité des applications à l'équipe des opérations cloud.

I

IaC

Considérez l'infrastructure comme un code.

politique basée sur l’identité

Politique attachée à un ou plusieurs principaux IAM qui définit leurs autorisations au sein de l' AWS Cloud environnement.

application inactive

Application dont l'utilisation moyenne du processeur et de la mémoire se situe entre 5 et 20 % sur une période de 90 jours. Dans un projet de migration, il est courant de retirer ces applications ou de les retenir sur site.

IIoT

Voir Internet industriel des objets.

infrastructure immuable

Modèle qui déploie une nouvelle infrastructure pour les charges de travail de production au lieu de mettre à jour, d'appliquer des correctifs ou de modifier l'infrastructure existante. Les infrastructures immuables sont intrinsèquement plus cohérentes, fiables et prévisibles que les infrastructures mutables. Pour plus d'informations, consultez les meilleures pratiques de déploiement à l'aide d'une infrastructure immuable dans le AWS Well-Architected Framework.

VPC entrant (d'entrée)

Dans une architecture AWS multi-comptes, un VPC qui accepte, inspecte et achemine les connexions réseau depuis l'extérieur d'une application. L'architecture de référence de sécuritéAWS recommande de configurer votre compte réseau avec des VPC entrants, sortants et d'inspection afin de protéger l'interface bidirectionnelle entre votre application et Internet en général.

migration incrémentielle

Stratégie de basculement dans le cadre de laquelle vous migrez votre application par petites parties au lieu d'effectuer un basculement complet unique. Par exemple, il se peut que vous ne transfériez que quelques microservices ou utilisateurs vers le nouveau système dans un premier temps. Après avoir vérifié que tout fonctionne correctement, vous pouvez transférer progressivement des microservices ou des utilisateurs supplémentaires jusqu'à ce que vous puissiez mettre hors service votre système hérité. Cette stratégie réduit les risques associés aux migrations de grande ampleur.

Industry 4.0

Terme introduit par Klaus Schwab en 2016 pour désigner la modernisation des processus de fabrication grâce aux avancées en matière de connectivité, de données en temps réel, d'automatisation, d'analyse et d'IA/ML.

infrastructure

Ensemble des ressources et des actifs contenus dans l'environnement d'une application.

infrastructure en tant que code (IaC)

Processus de mise en service et de gestion de l'infrastructure d'une application via un ensemble de fichiers de configuration. IaC est conçue pour vous aider à centraliser la gestion de l'infrastructure, à normaliser les ressources et à mettre à l'échelle rapidement afin que les nouveaux environnements soient reproductibles, fiables et cohérents.

internet industriel des objets (IIoT)

L'utilisation de capteurs et d'appareils connectés à Internet dans les secteurs industriels tels que la fabrication, l'énergie, l'automobile, les soins de santé, les sciences de la vie et l'agriculture. Pour plus d'informations, veuillez consulter Building an industrial Internet of Things (IIoT) digital transformation strategy.

VPC d'inspection

Dans une architecture AWS multi-comptes, un VPC centralisé qui gère les inspections du trafic réseau entre les VPC (identiques ou Régions AWS différents), Internet et les réseaux sur site. L'architecture de référence de sécuritéAWS recommande de configurer votre compte réseau avec des VPC entrants, sortants et d'inspection afin de protéger l'interface bidirectionnelle entre votre application et Internet en général.

Internet des objets (IoT)

Réseau d'objets physiques connectés dotés de capteurs ou de processeurs intégrés qui communiquent avec d'autres appareils et systèmes via Internet ou via un réseau de communication local. Pour plus d'informations, veuillez consulter la section Qu'est-ce que l'IoT ?.

interprétabilité

Caractéristique d'un modèle de machine learning qui décrit dans quelle mesure un être humain peut comprendre comment les prédictions du modèle dépendent de ses entrées. Pour plus d'informations, veuillez consulter Machine learning model interpretability with AWS.

IoT

Voir Internet des objets.

Bibliothèque d'informations informatiques (ITIL)

Ensemble de bonnes pratiques pour proposer des services informatiques et les aligner sur les exigences métier. L'ITIL constitue la base de l'ITSM.

gestion des services informatiques (ITSM)

Activités associées à la conception, à la mise en œuvre, à la gestion et à la prise en charge de services informatiques d'une organisation. Pour plus d'informations sur l'intégration des opérations cloud aux outils ITSM, veuillez consulter le guide d'intégration des opérations.

ITIL

Consultez la bibliothèque d'informations informatiques.

ITSM

Consultez la section Gestion des services informatiques.

L

contrôle d'accès basé sur des étiquettes (LBAC)

Une implémentation du contrôle d'accès obligatoire (MAC) dans laquelle une valeur d'étiquette de sécurité est explicitement attribuée aux utilisateurs et aux données elles-mêmes. L'intersection entre l'étiquette de sécurité utilisateur et l'étiquette de sécurité des données détermine les lignes et les colonnes visibles par l'utilisateur.

zone de destination

Une zone d'atterrissage est un AWS environnement multi-comptes bien conçu, évolutif et sécurisé. Il s'agit d'un point de départ à partir duquel vos entreprises peuvent rapidement lancer et déployer des charges de travail et des applications en toute confiance dans leur environnement de sécurité et d'infrastructure. Pour plus d'informations sur les zones de destination, veuillez consulter Setting up a secure and scalable multi-account AWS environment.

migration de grande envergure

Migration de 300 serveurs ou plus.

LBAC

Voir contrôle d'accès basé sur des étiquettes.

principe de moindre privilège

Bonne pratique de sécurité qui consiste à accorder les autorisations minimales nécessaires à l'exécution d'une tâche. Pour plus d'informations, veuillez consulter la rubrique Accorder les autorisations de moindre privilège dans la documentation IAM.

lift and shift

Voir 7 Rs.

système de poids faible

Système qui stocke d'abord l'octet le moins significatif. Voir aussi endianité.

environnements inférieurs

Voir environnement.

M

machine learning (ML)

Type d'intelligence artificielle qui utilise des algorithmes et des techniques pour la reconnaissance et l'apprentissage de modèles. Le ML analyse et apprend à partir de données enregistrées, telles que les données de l'Internet des objets (IoT), pour générer un modèle statistique basé sur des modèles. Pour plus d'informations, veuillez consulter Machine Learning.

branche principale

Voir la succursale.

malware

Logiciel conçu pour compromettre la sécurité ou la confidentialité de l'ordinateur. Les logiciels malveillants peuvent perturber les systèmes informatiques, divulguer des informations sensibles ou obtenir un accès non autorisé. Parmi les malwares, on peut citer les virus, les vers, les rançongiciels, les chevaux de Troie, les logiciels espions et les enregistreurs de frappe.

services gérés

Services AWS qui AWS gère la couche d'infrastructure, le système d'exploitation et les plateformes, et vous accédez aux points de terminaison pour stocker et récupérer des données.

Amazon Simple Storage Service (Amazon S3) et Amazon DynamoDB sont des exemples de services gérés. Ils sont également connus sous le nom de services abstraits.

système d'exécution de la fabrication (MES)

Un système logiciel pour le suivi, la surveillance, la documentation et le contrôle des processus de production qui convertissent les matières premières en produits finis dans l'atelier.

MAP

Voir Migration Acceleration Program.

mécanisme

Processus complet au cours duquel vous créez un outil, favorisez son adoption, puis inspectez les résultats afin de procéder aux ajustements nécessaires. Un mécanisme est un cycle qui se renforce et s'améliore lorsqu'il fonctionne. Pour plus d'informations, voir Création de mécanismes dans le cadre AWS Well-Architected.

compte membre

Tous, à l' Comptes AWS exception du compte de gestion, qui font partie d'une organisation dans AWS Organizations. Un compte ne peut être membre que d'une seule organisation à la fois.

MAILLES

Voir le système d'exécution de la fabrication.

Transport télémétrique en file d'attente de messages (MQTT)

Protocole de communication léger machine-to-machine (M2M), basé sur le modèle de publication/d'abonnement, pour les appareils IoT aux ressources limitées.

microservice

Petit service indépendant qui communique via des API bien définies et qui est généralement détenu par de petites équipes autonomes. Par exemple, un système d'assurance peut inclure des microservices qui mappent à des capacités métier, telles que les ventes ou le marketing, ou à des sous-domaines, tels que les achats, les réclamations ou l'analytique. Les avantages des microservices incluent l'agilité, la flexibilité de la mise à l'échelle, la facilité de déploiement, la réutilisation du code et la résilience. Pour plus d'informations, consultez la section Intégration de microservices à l'aide de services AWS sans serveur.

architecture de microservices

Approche de création d'une application avec des composants indépendants qui exécutent chaque processus d'application en tant que microservice. Ces microservices communiquent via une

interface bien définie à l'aide d'API légères. Chaque microservice de cette architecture peut être mis à jour, déployé et mis à l'échelle pour répondre à la demande de fonctions spécifiques d'une application. Pour plus d'informations, consultez la section Implémentation de microservices sur AWS.

Programme d'accélération des migrations (MAP)

Un AWS programme qui fournit un support de conseil, des formations et des services pour aider les entreprises à établir une base opérationnelle solide pour passer au cloud, et pour aider à compenser le coût initial des migrations. MAP inclut une méthodologie de migration pour exécuter les migrations héritées de manière méthodique, ainsi qu'un ensemble d'outils pour automatiser et accélérer les scénarios de migration courants.

migration à grande échelle

Processus consistant à transférer la majeure partie du portefeuille d'applications vers le cloud par vagues, un plus grand nombre d'applications étant déplacées plus rapidement à chaque vague. Cette phase utilise les bonnes pratiques et les enseignements tirés des phases précédentes pour implémenter une usine de migration d'équipes, d'outils et de processus en vue de rationaliser la migration des charges de travail grâce à l'automatisation et à la livraison agile. Il s'agit de la troisième phase de la stratégie de migration AWS.

usine de migration

Équipes interfonctionnelles qui rationalisent la migration des charges de travail grâce à des approches automatisées et agiles. Les équipes de Migration Factory comprennent généralement les opérations, les analystes commerciaux et les propriétaires, les ingénieurs de migration, les développeurs et les DevOps professionnels travaillant dans le cadre de sprints. Entre 20 et 50 % du portefeuille d'applications d'entreprise est constitué de modèles répétés qui peuvent être optimisés par une approche d'usine. Pour plus d'informations, veuillez consulter la rubrique discussion of migration factories et le guide Cloud Migration Factory dans cet ensemble de contenus.

métadonnées de migration

Informations relatives à l'application et au serveur nécessaires pour finaliser la migration. Chaque modèle de migration nécessite un ensemble de métadonnées de migration différent. Les exemples de métadonnées de migration incluent le sous-réseau cible, le groupe de sécurité et le AWS compte.

modèle de migration

Tâche de migration reproductible qui détaille la stratégie de migration, la destination de la migration et l'application ou le service de migration utilisé. Exemple : réorganisez la migration vers Amazon EC2 AWS avec le service de migration d'applications.

Évaluation du portefeuille de migration (MPA)

Outil en ligne qui fournit des informations pour valider l'analyse de rentabilisation en faveur de la migration vers le. AWS Cloud La MPA propose une évaluation détaillée du portefeuille (dimensionnement approprié des serveurs, tarification, comparaison du coût total de possession, analyse des coûts de migration), ainsi que la planification de la migration (analyse et collecte des données d'applications, regroupement des applications, priorisation des migrations et planification des vagues). L'outil MPA (connexion requise) est disponible gratuitement pour tous les AWS consultants et consultants APN Partner.

Évaluation de la préparation à la migration (MRA)

Processus qui consiste à obtenir des informations sur l'état de préparation d'une organisation au cloud, à identifier les forces et les faiblesses et à élaborer un plan d'action pour combler les lacunes identifiées, à l'aide du AWS CAF. Pour plus d'informations, veuillez consulter le guide de préparation à la migration. La MRA est la première phase de la stratégie de migration AWS.

stratégie de migration

L'approche utilisée pour migrer une charge de travail vers le AWS Cloud. Pour plus d'informations, reportez-vous aux 7 R de ce glossaire et à Mobiliser votre organisation pour accélérer les migrations à grande échelle.

ML

Voir apprentissage automatique.

modernisation

Transformation d'une application obsolète (héritée ou monolithique) et de son infrastructure en un système agile, élastique et hautement disponible dans le cloud afin de réduire les coûts, de gagner en efficacité et de tirer parti des innovations. Pour plus d'informations, consultez la section Stratégie de modernisation des applications dans le AWS Cloud.

évaluation de la préparation à la modernisation

Évaluation qui permet de déterminer si les applications d'une organisation sont prêtes à être modernisées, d'identifier les avantages, les risques et les dépendances, et qui détermine dans quelle mesure l'organisation peut prendre en charge l'état futur de ces applications. Le résultat

de l'évaluation est un plan de l'architecture cible, une feuille de route détaillant les phases de développement et les étapes du processus de modernisation, ainsi qu'un plan d'action pour combler les lacunes identifiées. Pour plus d'informations, consultez la section Évaluation de l'état de préparation à la modernisation des applications dans le AWS Cloud.

applications monolithiques (monolithes)

Applications qui s'exécutent en tant que service unique avec des processus étroitement couplés. Les applications monolithiques ont plusieurs inconvénients. Si une fonctionnalité de l'application connaît un pic de demande, l'architecture entière doit être mise à l'échelle. L'ajout ou l'amélioration des fonctionnalités d'une application monolithique devient également plus complexe lorsque la base de code s'élargit. Pour résoudre ces problèmes, vous pouvez utiliser une architecture de microservices. Pour plus d'informations, veuillez consulter Decomposing monoliths into microservices.

MPA

Voir Évaluation du portefeuille de migration.

MQTT

Voir Message Queuing Telemetry Transport.

classification multi-classes

Processus qui permet de générer des prédictions pour plusieurs classes (prédiction d'un résultat parmi plus de deux). Par exemple, un modèle de ML peut demander « Ce produit est-il un livre, une voiture ou un téléphone ? » ou « Quelle catégorie de produits intéresse le plus ce client ? ».

infrastructure mutable

Modèle qui met à jour et modifie l'infrastructure existante pour les charges de travail de production. Pour améliorer la cohérence, la fiabilité et la prévisibilité, le AWS Well-Architected Framework recommande l'utilisation d'une infrastructure immuable comme meilleure pratique.

O

OAC

Voir Contrôle d'accès à l'origine.

OAI

Voir l'identité d'accès à l'origine.

OCM

Voir gestion du changement organisationnel.

migration hors ligne

Méthode de migration dans laquelle la charge de travail source est supprimée au cours du processus de migration. Cette méthode implique un temps d'arrêt prolongé et est généralement utilisée pour de petites charges de travail non critiques.

OI

Consultez la section Intégration des opérations.

OLA

Voir l'accord au niveau opérationnel.

migration en ligne

Méthode de migration dans laquelle la charge de travail source est copiée sur le système cible sans être mise hors ligne. Les applications connectées à la charge de travail peuvent continuer à fonctionner pendant la migration. Cette méthode implique un temps d'arrêt nul ou minimal et est généralement utilisée pour les charges de travail de production critiques.

OPC-UA

Voir Open Process Communications - Architecture unifiée.

Communications par processus ouvert - Architecture unifiée (OPC-UA)

Un protocole de communication machine-to-machine (M2M) pour l'automatisation industrielle. L'OPC-UA fournit une norme d'interopérabilité avec des schémas de cryptage, d'authentification et d'autorisation des données.

accord au niveau opérationnel (OLA)

Accord qui précise ce que les groupes informatiques fonctionnels s'engagent à fournir les uns aux autres, afin de prendre en charge un contrat de niveau de service (SLA).

examen de l'état de préparation opérationnelle (ORR)

Une liste de questions et de bonnes pratiques associées qui vous aident à comprendre, évaluer, prévenir ou réduire l'ampleur des incidents et des défaillances possibles. Pour plus d'informations, voir Operational Readiness Reviews (ORR) dans le AWS Well-Architected Framework.

technologie opérationnelle (OT)

Systèmes matériels et logiciels qui fonctionnent avec l'environnement physique pour contrôler les opérations, les équipements et les infrastructures industriels. Dans le secteur manufacturier, l'intégration des systèmes OT et des technologies de l'information (IT) est au cœur des transformations de l'industrie 4.0.

intégration des opérations (OI)

Processus de modernisation des opérations dans le cloud, qui implique la planification de la préparation, l'automatisation et l'intégration. Pour en savoir plus, veuillez consulter le guide d'intégration des opérations.

journal de suivi d'organisation

Un parcours créé par AWS CloudTrail qui enregistre tous les événements pour tous les membres Comptes AWS d'une organisation dans AWS Organizations. Ce journal de suivi est créé dans chaque Compte AWS qui fait partie de l'organisation et suit l'activité de chaque compte. Pour plus d'informations, consultez la section Création d'un suivi pour une organisation dans la CloudTrail documentation.

gestion du changement organisationnel (OCM)

Cadre pour gérer les transformations métier majeures et perturbatrices du point de vue des personnes, de la culture et du leadership. L'OCM aide les organisations à se préparer et à effectuer la transition vers de nouveaux systèmes et de nouvelles politiques en accélérant l'adoption des changements, en abordant les problèmes de transition et en favorisant des changements culturels et organisationnels. Dans la stratégie de AWS migration, ce cadre est appelé accélération du personnel, en raison de la rapidité du changement requise dans les projets d'adoption du cloud. Pour plus d'informations, veuillez consulter le guide OCM.

contrôle d'accès d'origine (OAC)

Dans CloudFront, une option améliorée pour restreindre l'accès afin de sécuriser votre contenu Amazon Simple Storage Service (Amazon S3). L'OAC prend en charge tous les compartiments S3 dans leur ensemble Régions AWS, le chiffrement côté serveur avec AWS KMS (SSE-KMS) et les requêtes dynamiques `PUT` adressées au compartiment S3. `DELETE`

identité d'accès d'origine (OAI)

Dans CloudFront, une option permettant de restreindre l'accès afin de sécuriser votre contenu Amazon S3. Lorsque vous utilisez OAI, il CloudFront crée un principal auprès duquel Amazon S3 peut s'authentifier. Les principaux authentifiés ne peuvent accéder au contenu d'un compartiment

S3 que par le biais d'une distribution spécifique CloudFront . Voir également OAC, qui fournit un contrôle d'accès plus précis et amélioré.

OU

Voir l'examen de l'état de préparation opérationnelle.

DE

Voir technologie opérationnelle.

VPC sortant (de sortie)

Dans une architecture AWS multi-comptes, un VPC qui gère les connexions réseau initiées depuis une application. L'architecture de référence de sécuritéAWS recommande de configurer votre compte réseau avec des VPC entrants, sortants et d'inspection afin de protéger l'interface bidirectionnelle entre votre application et Internet en général.

P

limite des autorisations

Politique de gestion IAM attachée aux principaux IAM pour définir les autorisations maximales que peut avoir l'utilisateur ou le rôle. Pour plus d'informations, veuillez consulter la rubrique Limites des autorisations dans la documentation IAM.

informations personnelles identifiables (PII)

Informations qui, lorsqu'elles sont consultées directement ou associées à d'autres données connexes, peuvent être utilisées pour déduire raisonnablement l'identité d'une personne. Les exemples d'informations personnelles incluent les noms, les adresses et les informations de contact.

PII

Voir les informations personnelles identifiables.

manuel stratégique

Ensemble d'étapes prédéfinies qui capturent le travail associé aux migrations, comme la fourniture de fonctions d'opérations de base dans le cloud. Un manuel stratégique peut revêtir la forme de scripts, de runbooks automatisés ou d'un résumé des processus ou des étapes nécessaires au fonctionnement de votre environnement modernisé.

PLC

Voir contrôleur logique programmable.

PLM

Consultez la section Gestion du cycle de vie des produits.

politique

Objet capable de définir les autorisations (voir la politique basée sur l'identité), de spécifier les conditions d'accès (voir la politique basée sur les ressources) ou de définir les autorisations maximales pour tous les comptes d'une organisation dans AWS Organizations (voir la politique de contrôle des services).

persistance polyglotte

Choix indépendant de la technologie de stockage de données d'un microservice en fonction des modèles d'accès aux données et d'autres exigences. Si vos microservices utilisent la même technologie de stockage de données, ils peuvent rencontrer des difficultés d'implémentation ou présenter des performances médiocres. Les microservices sont plus faciles à mettre en œuvre, atteignent de meilleures performances, ainsi qu'une meilleure capacité de mise à l'échelle s'ils utilisent l'entrepôt de données le mieux adapté à leurs besoins. Pour plus d'informations, veuillez consulter Enabling data persistence in microservices.

évaluation du portefeuille

Processus de découverte, d'analyse et de priorisation du portefeuille d'applications afin de planifier la migration. Pour plus d'informations, veuillez consulter Evaluating migration readiness.

predicate

Une condition de requête qui renvoie `true` ou`false`, généralement située dans une `WHERE` clause.

prédicat pushdown

Technique d'optimisation des requêtes de base de données qui filtre les données de la requête avant le transfert. Cela réduit la quantité de données qui doivent être extraites et traitées à partir de la base de données relationnelle et améliore les performances des requêtes.

contrôle préventif

Contrôle de sécurité conçu pour empêcher qu'un événement ne se produise. Ces contrôles constituent une première ligne de défense pour empêcher tout accès non autorisé ou toute

modification indésirable de votre réseau. Pour plus d'informations, veuillez consulter Preventative controls dans Implementing security controls on AWS.

principal

Entité AWS capable d'effectuer des actions et d'accéder aux ressources. Cette entité est généralement un utilisateur root pour un Compte AWS rôle IAM ou un utilisateur. Pour plus d'informations, veuillez consulter la rubrique Principal dans Termes et concepts relatifs aux rôles, dans la documentation IAM.

Confidentialité dès la conception

Une approche de l'ingénierie des systèmes qui prend en compte la confidentialité tout au long du processus d'ingénierie.

zones hébergées privées

Conteneur qui contient des informations concernant la façon dont vous souhaitez qu'Amazon Route 53 réponde aux requêtes DNS pour un domaine et ses sous-domaines dans un ou plusieurs VPC. Pour plus d'informations, veuillez consulter Working with private hosted zones dans la documentation Route 53.

contrôle proactif

Contrôle de sécurité conçu pour empêcher le déploiement de ressources non conformes. Ces contrôles analysent les ressources avant qu'elles ne soient provisionnées. Si la ressource n'est pas conforme au contrôle, elle n'est pas provisionnée. Pour plus d'informations, consultez le guide de référence sur les contrôles dans la AWS Control Tower documentation et consultez la section Contrôles proactifs dans Implémentation des contrôles de sécurité sur AWS.

gestion du cycle de vie des produits (PLM)

Gestion des données et des processus d'un produit tout au long de son cycle de vie, depuis la conception, le développement et le lancement, en passant par la croissance et la maturité, jusqu'au déclin et au retrait.

environnement de production

Voir environnement.

contrôleur logique programmable (PLC)

Dans le secteur manufacturier, un ordinateur hautement fiable et adaptable qui surveille les machines et automatise les processus de fabrication.

pseudonymisation

Processus de remplacement des identifiants personnels dans un ensemble de données par des valeurs fictives. La pseudonymisation peut contribuer à protéger la vie privée. Les données pseudonymisées sont toujours considérées comme des données personnelles.

publier/souscrire (pub/sub)

Modèle qui permet des communications asynchrones entre les microservices afin d'améliorer l'évolutivité et la réactivité. Par exemple, dans un MES basé sur des microservices, un microservice peut publier des messages d'événements sur un canal auquel d'autres microservices peuvent s'abonner. Le système peut ajouter de nouveaux microservices sans modifier le service de publication.

Q

plan de requête

Série d'étapes, telles que des instructions, utilisées pour accéder aux données d'un système de base de données relationnelle SQL.

régression du plan de requêtes

Le cas où un optimiseur de service de base de données choisit un plan moins optimal qu'avant une modification donnée de l'environnement de base de données. Cela peut être dû à des changements en termes de statistiques, de contraintes, de paramètres d'environnement, de liaisons de paramètres de requêtes et de mises à jour du moteur de base de données.

R

Matrice RACI

Voir responsable, responsable, consulté, informé (RACI).

rançongiciel

Logiciel malveillant conçu pour bloquer l'accès à un système informatique ou à des données jusqu'à ce qu'un paiement soit effectué.

Matrice RASCI

Voir responsable, responsable, consulté, informé (RACI).

RCAC

Voir contrôle d'accès aux lignes et aux colonnes.

réplica en lecture

Copie d'une base de données utilisée en lecture seule. Vous pouvez acheminer les requêtes vers le réplica de lecture pour réduire la charge sur votre base de données principale.

réarchitecte

Voir 7 Rs.

objectif de point de récupération (RPO)

Durée maximale acceptable depuis le dernier point de récupération des données. Cela permet de déterminer ce qui est considéré comme une perte de données acceptable entre le dernier point de restauration et l'interruption du service.

objectif de temps de récupération (RTO)

Le délai maximum acceptable entre l'interruption du service et le rétablissement du service.

refactoriser

Voir 7 Rs.

Région

Un ensemble de AWS ressources dans une zone géographique. Chacune Région AWS est isolée et indépendante des autres pour garantir tolérance aux pannes, stabilité et résilience. Pour plus d'informations, voir Spécifier ce que Régions AWS votre compte peut utiliser.

régression

Technique de ML qui prédit une valeur numérique. Par exemple, pour résoudre le problème « Quel sera le prix de vente de cette maison ? », un modèle de ML pourrait utiliser un modèle de régression linéaire pour prédire le prix de vente d'une maison sur la base de faits connus à son sujet (par exemple, la superficie en mètres carrés).

réhéberger

Voir 7 Rs.

version

Dans un processus de déploiement, action visant à promouvoir les modifications apportées à un environnement de production.

déplacer

Voir 7 Rs.

replateforme

Voir 7 Rs.

rachat

Voir 7 Rs.

résilience

La capacité d'une application à résister aux perturbations ou à s'en remettre. La haute disponibilité et la reprise après sinistre sont des considérations courantes lors de la planification de la résilience dans le AWS Cloud. Pour plus d'informations, consultez AWS Cloud Résilience.

politique basée sur les ressources

Politique attachée à une ressource, comme un compartiment Amazon S3, un point de terminaison ou une clé de chiffrement. Ce type de politique précise les principaux auxquels l'accès est autorisé, les actions prises en charge et toutes les autres conditions qui doivent être remplies.

matrice responsable, redevable, consulté et informé (RACI)

Une matrice qui définit les rôles et les responsabilités de toutes les parties impliquées dans les activités de migration et les opérations cloud. Le nom de la matrice est dérivé des types de responsabilité définis dans la matrice : responsable (R), responsable (A), consulté (C) et informé (I). Le type de support (S) est facultatif. Si vous incluez le support, la matrice est appelée matrice RASCI, et si vous l'excluez, elle est appelée matrice RACI.

contrôle réactif

Contrôle de sécurité conçu pour permettre de remédier aux événements indésirables ou aux écarts par rapport à votre référence de sécurité. Pour plus d'informations, veuillez consulter la rubrique Responsive controls dans Implementing security controls on AWS.

retain

Voir 7 Rs.

se retirer

Voir 7 Rs.

rotation

Processus de mise à jour périodique d'un secret pour empêcher un attaquant d'accéder aux informations d'identification.

contrôle d'accès aux lignes et aux colonnes (RCAC)

Utilisation d'expressions SQL simples et flexibles dotées de règles d'accès définies. Le RCAC comprend des autorisations de ligne et des masques de colonnes.

RPO

Voir l'objectif du point de récupération.

RTO

Voir l'objectif en matière de temps de rétablissement.

runbook

Ensemble de procédures manuelles ou automatisées nécessaires à l'exécution d'une tâche spécifique. Elles visent généralement à rationaliser les opérations ou les procédures répétitives présentant des taux d'erreur élevés.

S

SAML 2.0

Un standard ouvert utilisé par de nombreux fournisseurs d'identité (IdPs). Cette fonctionnalité permet l'authentification unique fédérée (SSO), afin que les utilisateurs puissent se connecter AWS Management Console ou appeler les opérations d' AWS API sans que vous ayez à créer un utilisateur dans IAM pour tous les membres de votre organisation. Pour plus d'informations sur la fédération SAML 2.0, veuillez consulter À propos de la fédération SAML 2.0 dans la documentation IAM.

SCADA

Voir Contrôle de supervision et acquisition de données.

SCP

Voir la politique de contrôle des services.

secret

Dans AWS Secrets Manager des informations confidentielles ou restreintes, telles qu'un mot de passe ou des informations d'identification utilisateur, que vous stockez sous forme cryptée. Il comprend la valeur secrète et ses métadonnées. La valeur secrète peut être binaire, une chaîne unique ou plusieurs chaînes. Pour plus d'informations, voir Que contient le secret d'un Secrets Manager ? dans la documentation de Secrets Manager.

contrôle de sécurité

Barrière de protection technique ou administrative qui empêche, détecte ou réduit la capacité d'un assaillant d'exploiter une vulnérabilité de sécurité. Il existe quatre principaux types de contrôles de sécurité : préventifs, détectifs, réactifs et proactifs.

renforcement de la sécurité

Processus qui consiste à réduire la surface d'attaque pour la rendre plus résistante aux attaques. Cela peut inclure des actions telles que la suppression de ressources qui ne sont plus requises, la mise en œuvre des bonnes pratiques de sécurité consistant à accorder le moindre privilège ou la désactivation de fonctionnalités inutiles dans les fichiers de configuration.

système de gestion des informations et des événements de sécurité (SIEM)

Outils et services qui associent les systèmes de gestion des informations de sécurité (SIM) et de gestion des événements de sécurité (SEM). Un système SIEM collecte, surveille et analyse les données provenant de serveurs, de réseaux, d'appareils et d'autres sources afin de détecter les menaces et les failles de sécurité, mais aussi de générer des alertes.

automatisation des réponses de sécurité

Action prédéfinie et programmée conçue pour répondre automatiquement à un événement de sécurité ou y remédier. Ces automatisations servent de contrôles de sécurité détectifs ou réactifs qui vous aident à mettre en œuvre les meilleures pratiques AWS de sécurité. Parmi les actions de réponse automatique, citons la modification d'un groupe de sécurité VPC, l'application de correctifs à une instance Amazon EC2 ou la rotation des informations d'identification.

chiffrement côté serveur

Chiffrement des données à destination, par celui Service AWS qui les reçoit.

Politique de contrôle des services (SCP)

Politique qui propose un contrôle centralisé des autorisations pour tous les comptes d'une organisation dans AWS Organizations. Les SCP définissent des barrières de protection ou des

limites aux actions qu'un administrateur peut déléguer à des utilisateurs ou à des rôles. Vous pouvez utiliser les SCP comme listes d'autorisation ou de refus, pour indiquer les services ou les actions autorisés ou interdits. Pour plus d'informations, consultez la section Politiques de contrôle des services dans la AWS Organizations documentation.

point de terminaison du service

URL du point d'entrée pour un Service AWS. Pour vous connecter par programmation au service cible, vous pouvez utiliser un point de terminaison. Pour plus d'informations, veuillez consulter la rubrique Service AWS endpoints dans Références générales AWS.

contrat de niveau de service (SLA)

Accord qui précise ce qu'une équipe informatique promet de fournir à ses clients, comme le temps de disponibilité et les performances des services.

indicateur de niveau de service (SLI)

Mesure d'un aspect des performances d'un service, tel que son taux d'erreur, sa disponibilité ou son débit.

objectif de niveau de service (SLO)

Mesure cible qui représente l'état d'un service, tel que mesuré par un indicateur de niveau de service.

modèle de responsabilité partagée

Un modèle décrivant la responsabilité que vous partagez en matière AWS de sécurité et de conformité dans le cloud. AWS est responsable de la sécurité du cloud, alors que vous êtes responsable de la sécurité dans le cloud. Pour de plus amples informations, veuillez consulter Modèle de responsabilité partagée.

SIEM

Consultez les informations de sécurité et le système de gestion des événements.

point de défaillance unique (SPOF)

Défaillance d'un seul composant critique d'une application susceptible de perturber le système.

SLA

Voir le contrat de niveau de service.

SLI

Voir l'indicateur de niveau de service.

SLO

Voir l'objectif de niveau de service.

split-and-seed modèle

Modèle permettant de mettre à l'échelle et d'accélérer les projets de modernisation. Au fur et à mesure que les nouvelles fonctionnalités et les nouvelles versions de produits sont définies, l'équipe principale se divise pour créer des équipes de produit. Cela permet de mettre à l'échelle les capacités et les services de votre organisation, d'améliorer la productivité des développeurs et de favoriser une innovation rapide. Pour plus d'informations, consultez la section Approche progressive de la modernisation des applications dans le. AWS Cloud

SPOF

Voir point de défaillance unique.

schéma en étoile

Structure organisationnelle de base de données qui utilise une grande table de faits pour stocker les données transactionnelles ou mesurées et utilise une ou plusieurs tables dimensionnelles plus petites pour stocker les attributs des données. Cette structure est conçue pour être utilisée dans un entrepôt de données ou à des fins de business intelligence.

modèle de figuier étrangleur

Approche de modernisation des systèmes monolithiques en réécrivant et en remplaçant progressivement les fonctionnalités du système jusqu'à ce que le système hérité puisse être mis hors service. Ce modèle utilise l'analogie d'un figuier de vigne qui se développe dans un arbre existant et qui finit par supplanter son hôte. Le schéma a été présenté par Martin Fowler comme un moyen de gérer les risques lors de la réécriture de systèmes monolithiques. Pour obtenir un exemple d'application de ce modèle, veuillez consulter Modernizing legacy Microsoft ASP.NET (ASMX) web services incrementally by using containers and Amazon API Gateway.

sous-réseau

Plage d'adresses IP dans votre VPC. Un sous-réseau doit se trouver dans une seule zone de disponibilité.

contrôle de supervision et acquisition de données (SCADA)

Dans le secteur manufacturier, un système qui utilise du matériel et des logiciels pour surveiller les actifs physiques et les opérations de production.

chiffrement symétrique

Algorithme de chiffrement qui utilise la même clé pour chiffrer et déchiffrer les données.

tests synthétiques

Tester un système de manière à simuler les interactions des utilisateurs afin de détecter les problèmes potentiels ou de surveiller les performances. Vous pouvez utiliser Amazon CloudWatch Synthetics pour créer ces tests.

T

balises

Des paires clé-valeur qui agissent comme des métadonnées pour organiser vos AWS ressources. Les balises peuvent vous aider à gérer, identifier, organiser, rechercher et filtrer des ressources. Pour plus d'informations, veuillez consulter la rubrique Balisage de vos AWS ressources.

variable cible

La valeur que vous essayez de prédire dans le cadre du ML supervisé. Elle est également qualifiée de variable de résultat. Par exemple, dans un environnement de fabrication, la variable cible peut être un défaut du produit.

liste de tâches

Outil utilisé pour suivre les progrès dans un runbook. Liste de tâches qui contient une vue d'ensemble du runbook et une liste des tâches générales à effectuer. Pour chaque tâche générale, elle inclut le temps estimé nécessaire, le propriétaire et l'avancement.

environnement de test

Voir environnement.

entraînement

Pour fournir des données à partir desquelles votre modèle de ML peut apprendre. Les données d'entraînement doivent contenir la bonne réponse. L'algorithme d'apprentissage identifie des modèles dans les données d'entraînement, qui mettent en correspondance les attributs des données d'entrée avec la cible (la réponse que vous souhaitez prédire). Il fournit un modèle de ML qui capture ces modèles. Vous pouvez alors utiliser le modèle de ML pour obtenir des prédictions sur de nouvelles données pour lesquelles vous ne connaissez pas la cible.

passerelle de transit

Hub de transit de réseau que vous pouvez utiliser pour relier vos VPC et vos réseaux sur site. Pour plus d'informations, voir Qu'est-ce qu'une passerelle de transit dans la AWS Transit Gateway documentation.

flux de travail basé sur jonction

Approche selon laquelle les développeurs génèrent et testent des fonctionnalités localement dans une branche de fonctionnalités, puis fusionnent ces modifications dans la branche principale. La branche principale est ensuite intégrée aux environnements de développement, de préproduction et de production, de manière séquentielle.

accès sécurisé

Accorder des autorisations à un service que vous spécifiez pour effectuer des tâches au sein de votre organisation AWS Organizations et dans ses comptes en votre nom. Le service de confiance crée un rôle lié au service dans chaque compte, lorsque ce rôle est nécessaire, pour effectuer des tâches de gestion à votre place. Pour plus d'informations, consultez la section Utilisation AWS Organizations avec d'autres AWS services dans la AWS Organizations documentation.

réglage

Pour modifier certains aspects de votre processus d'entraînement afin d'améliorer la précision du modèle de ML. Par exemple, vous pouvez entraîner le modèle de ML en générant un ensemble d'étiquetage, en ajoutant des étiquettes, puis en répétant ces étapes plusieurs fois avec différents paramètres pour optimiser le modèle.

équipe de deux pizzas

Une petite DevOps équipe que vous pouvez nourrir avec deux pizzas. Une équipe de deux pizzas garantit les meilleures opportunités de collaboration possible dans le développement de logiciels.

U

incertitude

Un concept qui fait référence à des informations imprécises, incomplètes ou inconnues susceptibles de compromettre la fiabilité des modèles de ML prédictifs. Il existe deux types d'incertitude : l'incertitude épistémique est causée par des données limitées et incomplètes, alors que l'incertitude aléatoire est causée par le bruit et le caractère aléatoire inhérents aux données.

Pour plus d'informations, veuillez consulter le guide Quantifying uncertainty in deep learning systems.

tâches indifférenciées

Également connu sous le nom de « levage de charges lourdes », ce travail est nécessaire pour créer et exploiter une application, mais qui n'apporte pas de valeur directe à l'utilisateur final ni d'avantage concurrentiel. Les exemples de tâches indifférenciées incluent l'approvisionnement, la maintenance et la planification des capacités.

environnements supérieurs

Voir environnement.

V

mise à vide

Opération de maintenance de base de données qui implique un nettoyage après des mises à jour incrémentielles afin de récupérer de l'espace de stockage et d'améliorer les performances.

contrôle de version

Processus et outils permettant de suivre les modifications, telles que les modifications apportées au code source dans un référentiel.

Appairage de VPC

Connexion entre deux VPC qui vous permet d'acheminer le trafic à l'aide d'adresses IP privées. Pour plus d'informations, veuillez consulter la rubrique Qu'est-ce que l'appairage de VPC ? dans la documentation Amazon VPC.

vulnérabilités

Défaut logiciel ou matériel qui compromet la sécurité du système.

W

cache actif

Cache tampon qui contient les données actuelles et pertinentes fréquemment consultées. L'instance de base de données peut lire à partir du cache tampon, ce qui est plus rapide que la lecture à partir de la mémoire principale ou du disque.

données chaudes

Données rarement consultées. Lorsque vous interrogez ce type de données, des requêtes modérément lentes sont généralement acceptables.

fonction de fenêtre

Fonction SQL qui effectue un calcul sur un groupe de lignes liées d'une manière ou d'une autre à l'enregistrement en cours. Les fonctions de fenêtre sont utiles pour traiter des tâches, telles que le calcul d'une moyenne mobile ou l'accès à la valeur des lignes en fonction de la position relative de la ligne en cours.

charge de travail

Ensemble de ressources et de code qui fournit une valeur métier, par exemple une application destinée au client ou un processus de backend.

flux de travail

Groupes fonctionnels d'un projet de migration chargés d'un ensemble de tâches spécifique. Chaque flux de travail est indépendant, mais prend en charge les autres flux de travail du projet. Par exemple, le flux de travail du portefeuille est chargé de prioriser les applications, de planifier les vagues et de collecter les métadonnées de migration. Le flux de travail du portefeuille fournit ces actifs au flux de travail de migration, qui migre ensuite les serveurs et les applications.

VER

Voir écrire une fois, lire plusieurs.

WQF

Consultez le cadre de qualification des charges de travail AWS.

écrire une fois, lire plusieurs (WORM)

Modèle de stockage qui écrit les données une seule fois et empêche leur suppression ou leur modification. Les utilisateurs autorisés peuvent lire les données autant de fois que nécessaire, mais ils ne peuvent pas les modifier. Cette infrastructure de stockage de données est considérée comme immuable.

Z

exploit Zero-Day

Une attaque, généralement un logiciel malveillant, qui tire parti d'une vulnérabilité de type « jour zéro ».

vulnérabilité de type « jour zéro »

Une faille ou une vulnérabilité non atténuée dans un système de production. Les acteurs malveillants peuvent utiliser ce type de vulnérabilité pour attaquer le système. Les développeurs prennent souvent conscience de la vulnérabilité à la suite de l'attaque.

application zombie

Application dont l'utilisation moyenne du processeur et de la mémoire est inférieure à 5 %. Dans un projet de migration, il est courant de retirer ces applications.

Les traductions sont fournies par des outils de traduction automatique. En cas de conflit entre le contenu d'une traduction et celui de la version originale en anglais, la version anglaise prévaudra.