Modelli di progettazione, architetture e implementazioni del cloud

AWS Guida prescrittiva

AWS Guida prescrittiva: Modelli di progettazione, architetture e implementazioni del cloud

Table of Contents

Modelli di progettazione, architetture e implementazioni del cloud

Anitha Deenadayalan, Amazon Web Services (AWS)

Maggio 2024 (storia del documento)

Questa guida fornisce linee guida per l'implementazione di modelli di progettazione di modernizzazione di uso comune utilizzando AWS i servizi. Un numero crescente di applicazioni moderne viene progettato utilizzando architetture di microservizi per raggiungere la scalabilità, migliorare la velocità di rilascio, ridurre la portata dell'impatto delle modifiche e ridurre la regressione. Ciò comporta una maggiore produttività degli sviluppatori e una maggiore agilità, una migliore innovazione e una maggiore attenzione alle esigenze aziendali. Le architetture a microservizi supportano anche l'uso della migliore tecnologia per il servizio e il database e promuovono il codice poliglotta e la persistenza poliglotta.

Tradizionalmente, le applicazioni monolitiche vengono eseguite in un unico processo, utilizzano un unico datastore e vengono eseguite su server dimensionabili verticalmente. Per contro, le moderne applicazioni di microservizi sono granulari, hanno domini di errore indipendenti, vengono eseguite come servizi sulla rete e possono utilizzare più di un datastore a seconda del caso d'uso. I servizi si dimensionano orizzontalmente e una singola transazione può estendersi su più database. I team di sviluppo devono concentrarsi sulla comunicazione di rete, sulla persistenza poliglotta, sul dimensionamento orizzontale, sulla coerenza finale e sulla gestione delle transazioni tra i datastore quando sviluppano applicazioni utilizzando architetture di microservizi. Pertanto, i modelli di modernizzazione sono fondamentali per risolvere i problemi più comuni nello sviluppo di applicazioni moderne e aiutano ad accelerare la distribuzione del software.

Questa guida fornisce un riferimento tecnico per architetti cloud, responsabili tecnici, titolari di applicazioni e aziende e sviluppatori che desiderano scegliere l'architettura cloud giusta per modelli di progettazione basati su best practice ben architettate. Ogni modello discusso in questa guida affronta uno o più scenari noti nelle architetture di microservizi. La guida discute i problemi e le considerazioni associati a ciascun modello, fornisce un'implementazione architettonica di alto livello e descrive l'implementazione AWS per il modello. Laddove disponibili, vengono forniti GitHub esempi open source e collegamenti ai workshop.

La guida copre i seguenti modelli:

- Livello anticorruzione
- Modelli di routing delle API:
  - Routing dei nomi host
  - Routing dei percorsi
  - Routing delle intestazioni HTTP
- Interruttore
- Approvvigionamento di eventi
- Architettura esagonale
- Pubblicazione-sottoscrizione
- Nuovo tentativo con backoff
- Modelli saga:
  - Coreografia saga
  - Orchestrazione saga
- Scatter-gather
- Strangler fig
- Posta in uscita transazionale

Obiettivi aziendali specifici

Utilizzando i modelli descritti in questa guida per modernizzare le applicazioni, è possibile:

- Progettare e implementare architetture affidabili, sicure ed efficienti dal punto di vista operativo, ottimizzate per costi e prestazioni.
- Ridurre il tempo di ciclo per i casi d'uso che richiedono questi modelli, in modo da poterti concentrare invece sulle sfide specifiche dell'organizzazione.
- Accelerare lo sviluppo standardizzando le implementazioni dei modelli utilizzando i servizi AWS.
- Aiutare i tuoi sviluppatori a creare applicazioni moderne senza ereditare debiti tecnici.

Schema a strati anticorruzione

Intento

Il pattern anticorruzione (ACL) funge da livello di mediazione che traduce la semantica del modello di dominio da un sistema all'altro. Traduce il modello del contesto limitato a monte (monolite) in un modello adatto al contesto limitato a valle (microservizio) prima di utilizzare il contratto di comunicazione stabilito dal team a monte. Questo modello può essere applicabile quando il contesto limitato a valle contiene un sottodominio principale o il modello a monte è un sistema legacy non modificabile. Riduce inoltre il rischio di trasformazione e le interruzioni dell'attività impedendo ai chiamanti di modificare le chiamate quando devono essere reindirizzate in modo trasparente al sistema di destinazione.

Motivazione

Durante il processo di migrazione, quando un'applicazione monolitica viene migrata nei microservizi, potrebbero verificarsi cambiamenti nella semantica del modello di dominio del servizio appena migrato. Quando sono necessarie le funzionalità all'interno del monolite per chiamare questi microservizi, le chiamate devono essere indirizzate al servizio migrato senza richiedere alcuna modifica ai servizi chiamanti. Il pattern ACL consente al monolite di chiamare i microservizi in modo trasparente agendo come un adattatore o uno strato di facciata che traduce le chiamate nella semantica più recente.

Applicabilità

Prendi in considerazione l'utilizzo di questo modello quando:

- L'applicazione monolitica esistente deve comunicare con una funzione che è stata migrata in un microservizio e il modello e la semantica del dominio del servizio migrati differiscono dalla funzionalità originale.
- Due sistemi hanno una semantica diversa e richiedono lo scambio di dati, ma non è pratico modificare un sistema per renderlo compatibile con l'altro sistema.
- Desiderate utilizzare un approccio rapido e semplificato per adattare un sistema all'altro con un impatto minimo.

- L'applicazione comunica con un sistema esterno.

Problemi e considerazioni

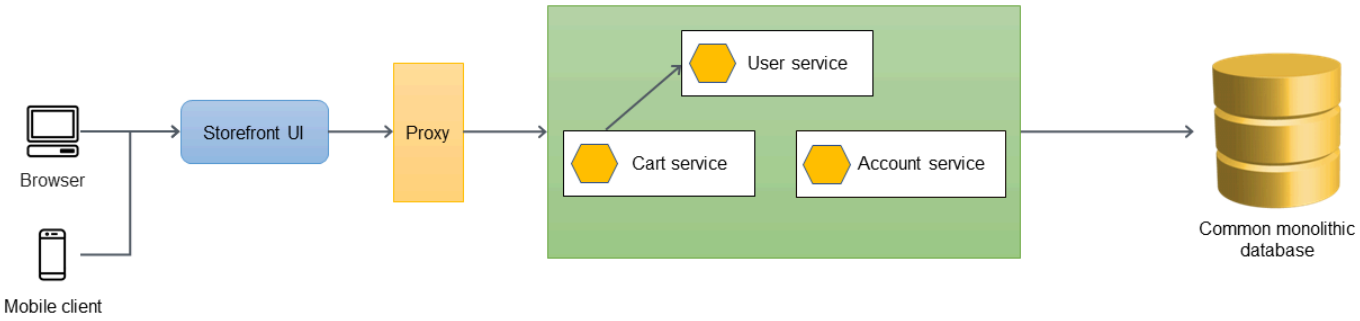
- Dipendenze del team:Quando diversi servizi in un sistema sono di proprietà di team diversi, la nuova semantica del modello di dominio nei servizi migrati può portare a cambiamenti nei sistemi di chiamata. Tuttavia, i team potrebbero non essere in grado di apportare queste modifiche in modo coordinato, perché potrebbero avere altre priorità. L'ACL separa i chiamanti e traduce le chiamate in modo che corrispondano alla semantica dei nuovi servizi, evitando così che i chiamanti debbano apportare modifiche al sistema attuale.
- Spese generali operative:Il modello ACL richiede uno sforzo aggiuntivo per il funzionamento e la manutenzione. Questo lavoro include l'integrazione dell'ACL con strumenti di monitoraggio e avviso, il processo di rilascio e i processi di integrazione continua e distribuzione continua (CI/CD).
- Singolo punto di errore:Qualsiasi errore nell'ACL può rendere irraggiungibile il servizio di destinazione, causando problemi alle applicazioni. Per mitigare questo problema, è necessario integrare funzionalità di ripetizione dei tentativi e interruttori automatici. Vedi lariprova con backoffeinterruttoremodelli per comprendere meglio queste opzioni. L'impostazione di avvisi e registri appropriati migliorerà il tempo medio di risoluzione (MTTR).
- Debito tecnico:Come parte della tua strategia di migrazione o modernizzazione, valuta se l'ACL sarà una soluzione transitoria o provvisoria o una soluzione a lungo termine. Se si tratta di una soluzione provvisoria, è necessario registrare l'ACL come debito tecnico e disattivarlo dopo la migrazione di tutti i chiamanti dipendenti.
- Latenza:Il livello aggiuntivo può introdurre latenza dovuta alla conversione delle richieste da un'interfaccia all'altra. Si consiglia di definire e testare la tolleranza delle prestazioni nelle applicazioni sensibili ai tempi di risposta prima di implementare ACL negli ambienti di produzione.
- Collo di bottiglia in scala:Nelle applicazioni ad alto carico in cui i servizi possono essere scalati in base al picco di carico, l'ACL può diventare un collo di bottiglia e causare problemi di scalabilità. Se il servizio di destinazione è scalabile su richiesta, è necessario progettare l'ACL in modo che sia scalabile di conseguenza.
- Implementazione specifica o condivisa del servizio:È possibile progettare ACL come oggetto condiviso per convertire e reindirizzare le chiamate a più servizi o classi specifiche del servizio. Prendi in considerazione la latenza, la scalabilità e la tolleranza agli errori quando determini il tipo di implementazione per ACL.

Implementazione

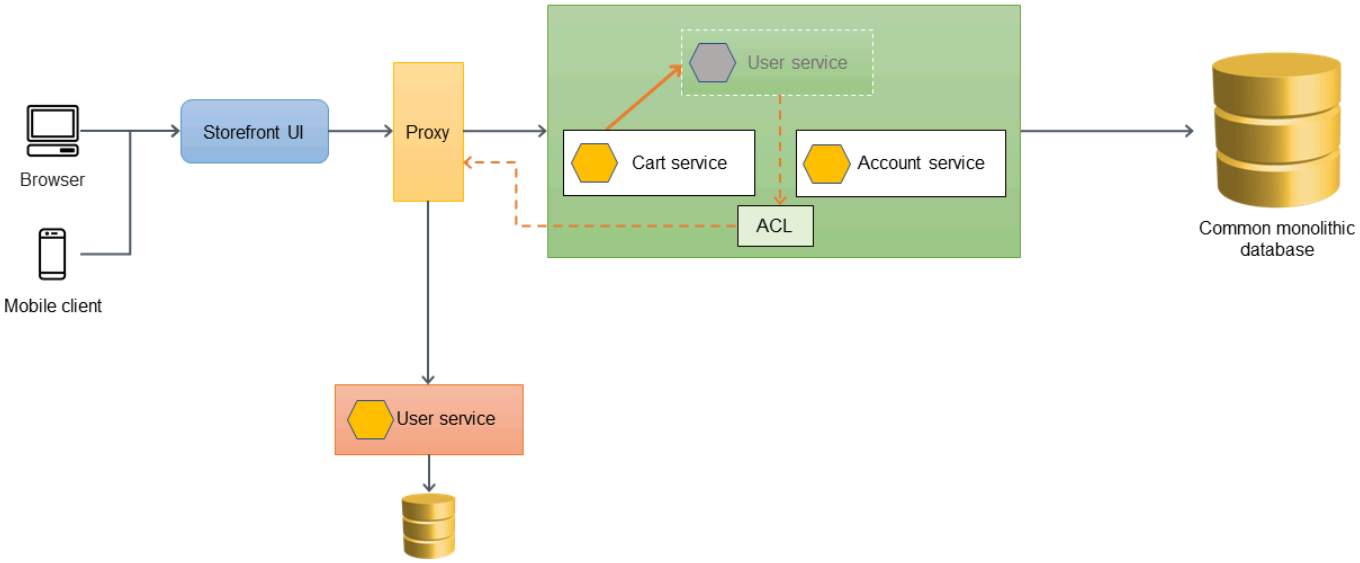
Puoi implementare ACL all'interno della tua applicazione monolitica come classe specifica per il servizio da migrare o come servizio indipendente. L'ACL deve essere disattivato dopo la migrazione di tutti i servizi dipendenti nell'architettura dei microservizi.

Architettura di alto livello

Nell'architettura di esempio seguente, un'applicazione monolitica dispone di tre servizi: servizio utente, servizio carrello e servizio account. Il servizio carrello dipende dal servizio utente e l'applicazione utilizza un database relazionale monolitico.

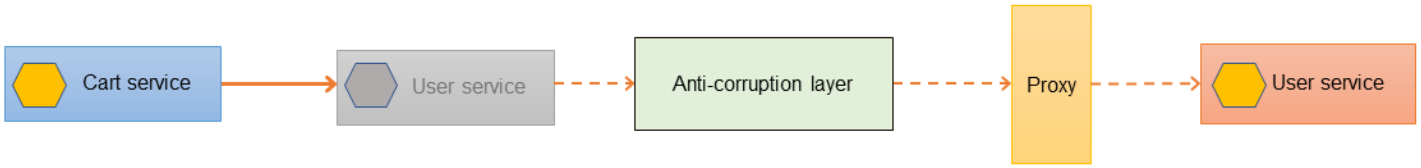


Nella seguente architettura, il servizio utente è stato migrato a un nuovo microservizio. Il servizio cart chiama il servizio utente, ma l'implementazione non è più disponibile all'interno del monolite. È anche probabile che l'interfaccia del servizio appena migrato non corrisponda all'interfaccia precedente, quando si trovava all'interno dell'applicazione monolitica.



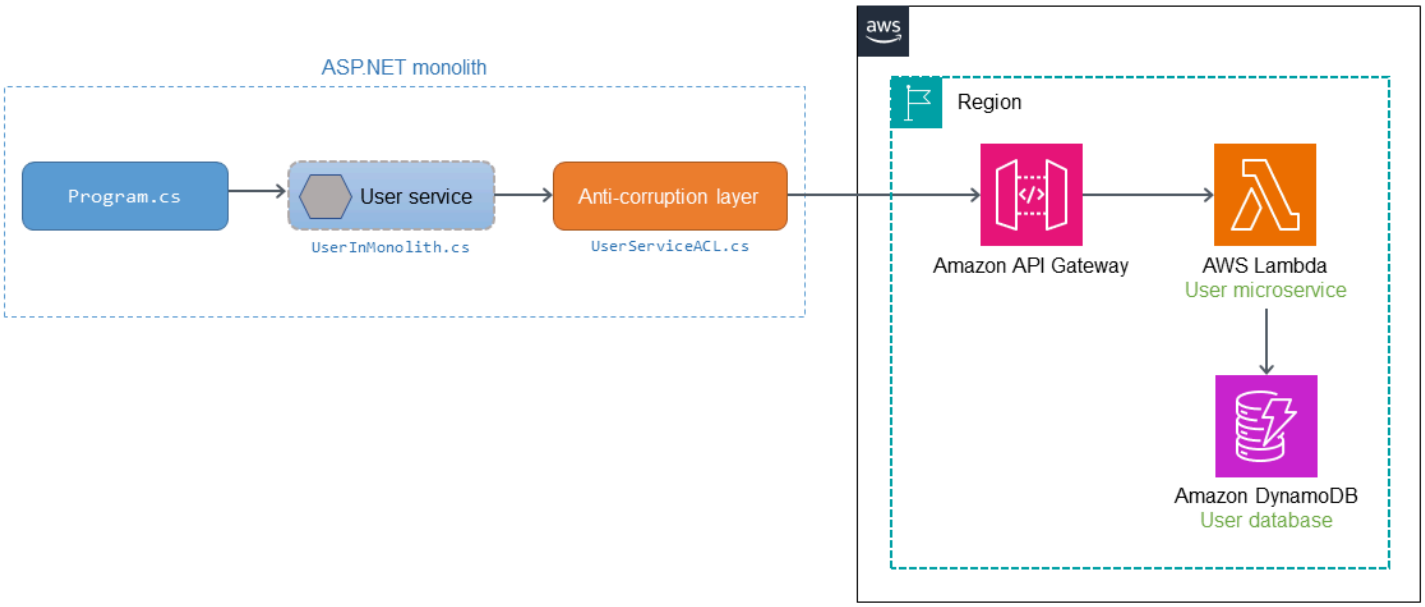
Se il servizio carrello deve chiamare direttamente il servizio utente appena migrato, ciò richiederà modifiche al servizio carrello e un test approfondito dell'applicazione monolitica. Ciò può aumentare il rischio di trasformazione e l'interruzione del business. L'obiettivo dovrebbe essere quello di ridurre al minimo le modifiche alla funzionalità esistente dell'applicazione monolitica.

In questo caso, ti consigliamo di introdurre un ACL tra il vecchio servizio utente e il servizio utente appena migrato. L'ACL funziona come un adattatore o una facciata che converte le chiamate nell'interfaccia più recente. L'ACL può essere implementato all'interno dell'applicazione monolitica come classe (ad esempio,`UserServiceFacadeo``UserServiceAdapter`) che è specifico per il servizio che è stato migrato. Il livello anticorruzione deve essere disattivato dopo la migrazione di tutti i servizi dipendenti nell'architettura dei microservizi.



Implementazione utilizzandoAWSservizi

Il diagramma seguente mostra come è possibile implementare questo esempio di ACL utilizzandoAWSservizi.



Il microservizio utente viene migrato dall'applicazione monolitica ASP.NET e distribuito comeAWS Lambdafunzione su AWS. Le chiamate alla funzione Lambda vengono indirizzate tramiteGateway

API Amazon. L'ACL viene implementato nel monolite per tradurre la chiamata per adattarla alla semantica del microservizio utente.

Quando`Program.cs`chiama il servizio utenti (`UserInMonolith.cs`) all'interno del monolite, la chiamata viene indirizzata all'ACL (`UserServiceACL.cs`). L'ACL traduce la chiamata nella nuova semantica e interfaccia e chiama il microservizio tramite l'endpoint API Gateway. Il chiamante (`Program.cs`) non è a conoscenza della traduzione e del routing che avvengono nel servizio utenti e nell'ACL. Poiché il chiamante non è a conoscenza delle modifiche al codice, si riducono le interruzioni dell'attività e si riducono i rischi di trasformazione.

Codice di esempio

Il seguente frammento di codice fornisce le modifiche al servizio originale e l'implementazione di`UserServiceACL.cs`. Quando viene ricevuta una richiesta, il servizio utente originale chiama l'ACL. L'ACL converte l'oggetto di origine in modo che corrisponda all'interfaccia del servizio appena migrato, chiama il servizio e restituisce la risposta al chiamante.

```
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;
    }
}
```

GitHubmagazzino

Per un'implementazione completa dell'architettura di esempio per questo modello, vedereGitHubrepository pressohttps://github.com/aws-samples/anti-corruption-layer-pattern.

Contenuti correlati

- Motivo Strangler fig
- Schema dell'interruttore
- Riprova con schema di backoff

Modelli di routing delle API

In ambienti di sviluppo agili, i team autonomi (ad esempio squadre e tribù) possiedono uno o più servizi che includono molti microservizi. I team espongono questi servizi come API per consentire ai consumer di interagire con il loro gruppo di servizi e azioni.

Esistono tre metodi principali per esporre le API HTTP ai consumer upstream utilizzando nomi host e percorsi:

Metodo	Descrizione	Esempio
Routing dei nomi host	Esponi ogni servizio come hostname.	`billing.api.example.com`
Routing dei percorsi	Esponi ogni servizio come percorso.	`api.example.com/billing`
Routing basato su intestazione	Esponi ogni servizio come intestazione HTTP.	`x-example-action: something`

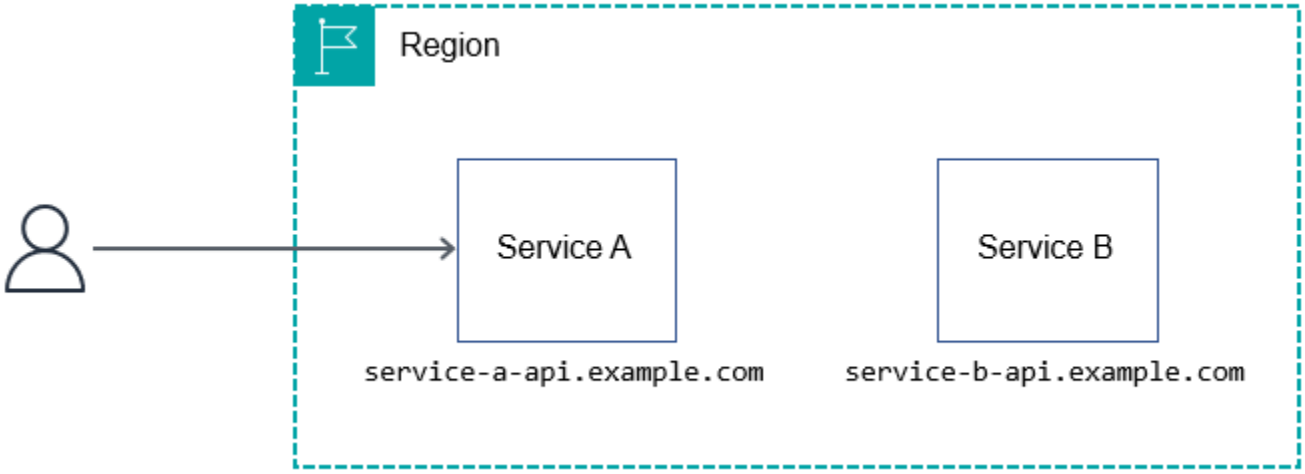
Questa sezione descrive i casi d'uso tipici di questi tre metodi di routing e i relativi compromessi per aiutarti a decidere quale metodo si adatta meglio alle tue esigenze e alla tua struttura organizzativa.

Modello di routing dei nomi host

Il routing per nome host è un meccanismo per isolare i servizi API assegnando a ciascuna API il proprio nome host; ad esempio, `service-a.api.example.com` o `service-a.example.com`.

Caso d'uso tipico

Il routing tramite i nomi host riduce l'attrito nelle release, perché nulla viene condiviso tra i team di assistenza. I team sono responsabili della gestione di tutto, dagli inserimenti DNS alle operazioni di servizio in produzione.



Pro

Il routing dei nomi host è di gran lunga il metodo più semplice e scalabile per il routing delle API HTTP. Puoi utilizzare qualsiasi AWS servizio pertinente per creare un'architettura che segua questo metodo: puoi creare un'architettura con Amazon API Gateway AWS AppSync, Application Load Balancers e Amazon Elastic Compute Cloud (Amazon EC2) o qualsiasi altro servizio conforme a HTTP.

I team possono utilizzare il routing dei nomi host per possedere completamente il proprio sottodominio. Inoltre, semplifica l'isolamento, il test e l'orchestrazione di distribuzioni per versioni o versioni specifiche, ad esempio o. Regioni AWS `region.service-a.api.example.com` `dev.region.service-a.api.example.com`

Contro

Quando utilizzi il routing dei nomi host, i consumer devono ricordare diversi nomi host per interagire con ogni API che si espone. Puoi mitigare questo problema fornendo un SDK per il client. Tuttavia, gli SDK dei client presentano una serie di problemi. Ad esempio, devono supportare aggiornamenti continui, più lingue, il controllo delle versioni, la comunicazione delle modifiche più importanti causate da problemi di sicurezza o correzioni di bug, la documentazione e così via.

Quando si utilizza il routing dei nomi host, è inoltre necessario registrare il sottodominio o il dominio ogni volta che si crea un nuovo servizio.

Modello di routing dei percorsi

Il routing dei percorsi è il meccanismo che raggruppa più o tutte le API con lo stesso nome host e utilizza un URI di richiesta per isolare i servizi, ad esempio, `api.example.com/service-a` o `api.example.com/service-b`.

Caso d'uso tipico

La maggior parte dei team sceglie questo metodo perché desidera un'architettura semplice: uno sviluppatore deve ricordare solo un URL, ad esempio `api.example.com` per interagire con l'API HTTP. La documentazione delle API è spesso più facile da consultare perché spesso è tenuta tutta insieme anziché essere suddivisa su diversi portali o PDF.

Il routing basato sui percorsi è considerato un meccanismo semplice per condividere un'API HTTP. Tuttavia, comporta un sovraccarico operativo come configurazione, autorizzazione, integrazioni e latenza aggiuntiva dovuta a più hop. Richiede inoltre processi avanzati di gestione delle modifiche per garantire che un'errata configurazione non interrompa tutti i servizi.

Sono disponibili diversi modi per condividere un'API e indirizzare in modo efficace al servizio corretto. AWS Le sezioni seguenti illustrano tre approcci: proxy inverso del servizio HTTP, API Gateway e Amazon CloudFront. Nessuno degli approcci suggeriti per l'unificazione dei servizi API si basa sui servizi downstream in esecuzione su AWS. I servizi possono essere eseguiti ovunque senza problemi o con qualsiasi tecnologia, purché siano compatibili con HTTP.

Proxy inverso del servizio HTTP

È possibile utilizzare un server HTTP come NGINX per creare configurazioni di routing dinamiche. In un'architettura Kubernetes, puoi anche creare una regola di ingresso per abbinare un percorso a un servizio. (Questa guida non copre l'accesso a Kubernetes; consulta la documentazione di Kubernetes per ulteriori informazioni.)

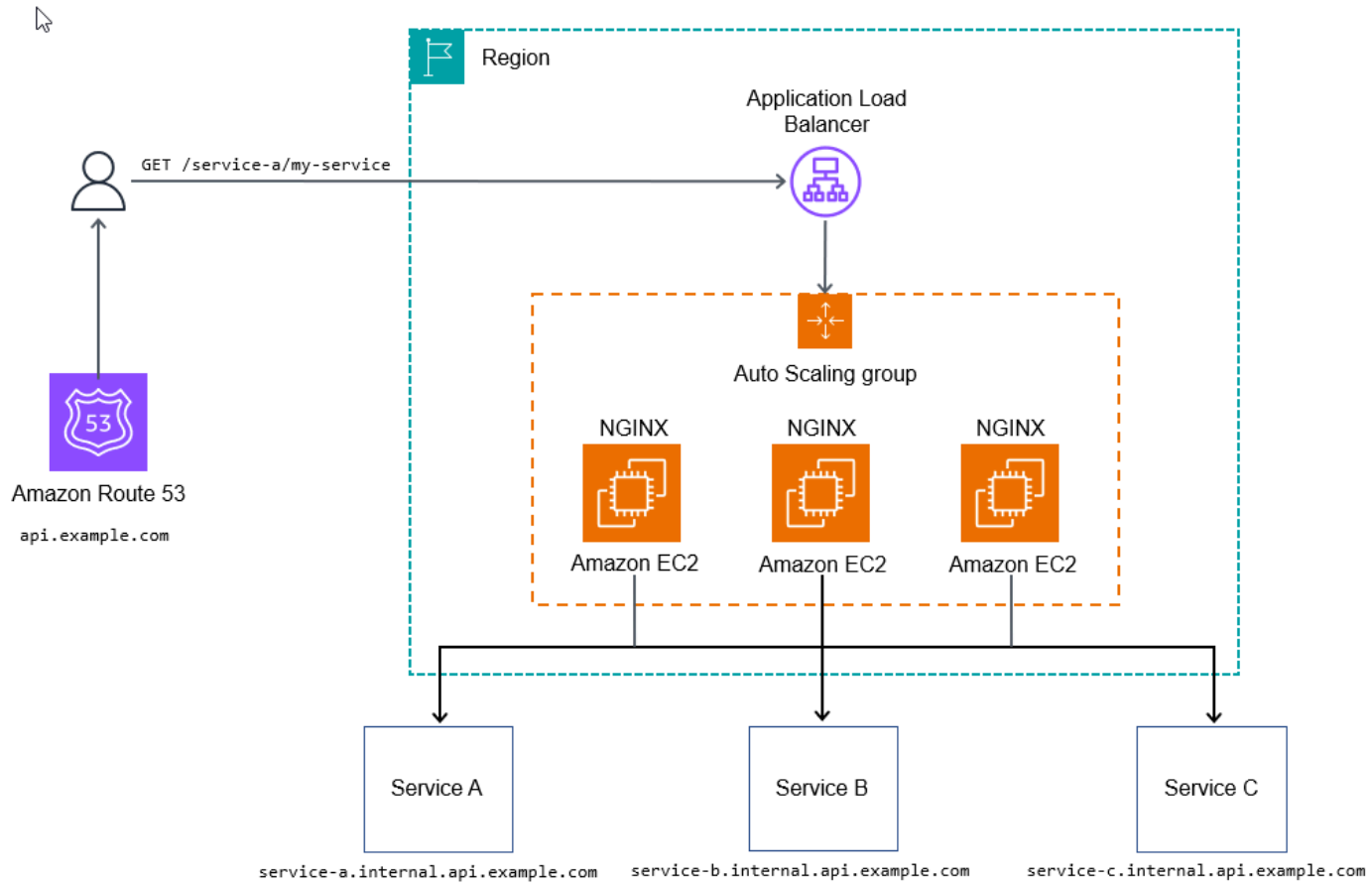
La seguente configurazione per NGINX mappa dinamicamente una richiesta HTTP di `api.example.com/my-service/` a `my-service.internal.api.example.com`.

```
server {
    listen  80;

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

```
}
```

Il diagramma seguente illustra il metodo proxy inverso del servizio HTTP.



Questo approccio potrebbe essere sufficiente per alcuni casi d'uso che non utilizzano configurazioni aggiuntive per avviare l'elaborazione delle richieste, consentendo all'API downstream di raccogliere parametri e log.

Per prepararti alla produzione operativa, ti consigliamo di aggiungere osservabilità a ogni livello dello stack, configurazioni aggiuntive o script per personalizzare il punto di ingresso dell'API e consentire funzionalità più avanzate come la limitazione della frequenza o i token di utilizzo.

Pro

L'obiettivo finale del metodo del proxy inverso del servizio HTTP è creare un approccio dimensionabile e gestibile per unificare le API in un unico dominio in modo che appaia coerente per qualsiasi consumer di API. Questo approccio consente inoltre ai team di assistenza di implementare

e gestire le proprie API, con un sovraccarico minimo dopo l'implementazione. AWS i servizi gestiti per la tracciabilità, come AWS X-Rayo AWS WAF, sono ancora applicabili qui.

Contro

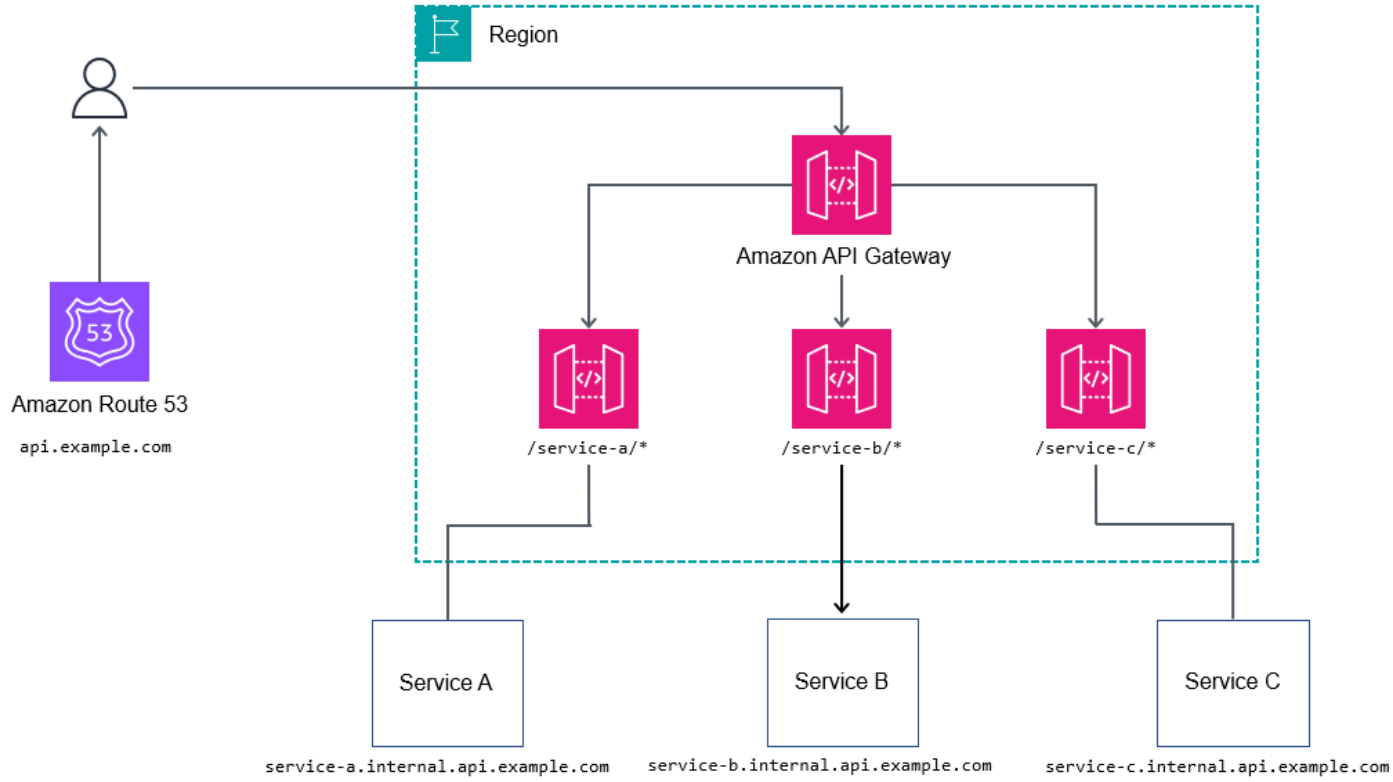
Il principale svantaggio di questo approccio è rappresentato dai numerosi test e dalla gestione dei componenti dell'infrastruttura necessari, anche se questo potrebbe non essere un problema se si dispone di team di ingegneria dell'affidabilità del sito (SRE).

Con questo metodo si verifica un punto critico in termini di costi. A volumi medio-bassi, è più costoso di alcuni degli altri metodi descritti in questa guida. A volumi elevati, è molto conveniente (circa 100.000 transazioni al secondo o più).

API Gateway

Il servizio Gateway Amazon API (REST API e API HTTP) può instradare il traffico in modo simile al metodo del proxy inverso del servizio HTTP. L'utilizzo di un gateway API in modalità proxy HTTP offre un modo semplice per inserire molti servizi in un punto di ingresso al sottodominio di primo livello `api.example.com` e quindi inoltrare le richieste al servizio nidificato, ad esempio `billing.internal.api.example.com`.

Probabilmente non vorrai usare troppe informazioni dettagliate mappando ogni percorso di ogni servizio nel gateway API principale o root. Scegli invece percorsi con caratteri jolly, ad esempio `/billing/*` per inoltrare le richieste al servizio di fatturazione. Evitando di mappare tutti i percorsi del gateway API principale o root, si ottiene una maggiore flessibilità rispetto alle API, poiché non è necessario aggiornare il gateway API principale a ogni modifica dell'API.



Pro

Per controllare flussi di lavoro più complessi, come la modifica degli attributi della richiesta, le REST API utilizzano il Velocity Template Language (VTL) di'Apache per consentire di modificare la richiesta e la risposta. Le REST API possono offrire vantaggi aggiuntivi come i seguenti:

- Autenticazione N/Z con AWS Identity and Access Management (IAM), Amazon Cognito o autorizzatori AWS Lambda
- AWS X-Ray per tracciare
- Integrazione con AWS WAF
- Limitazione della velocità di base
- Token di utilizzo per raggruppare i consumer in diversi livelli (consulta Richieste API Throttle per una migliore velocità di trasmissione effettiva nella documentazione di Gateway API)

Contro

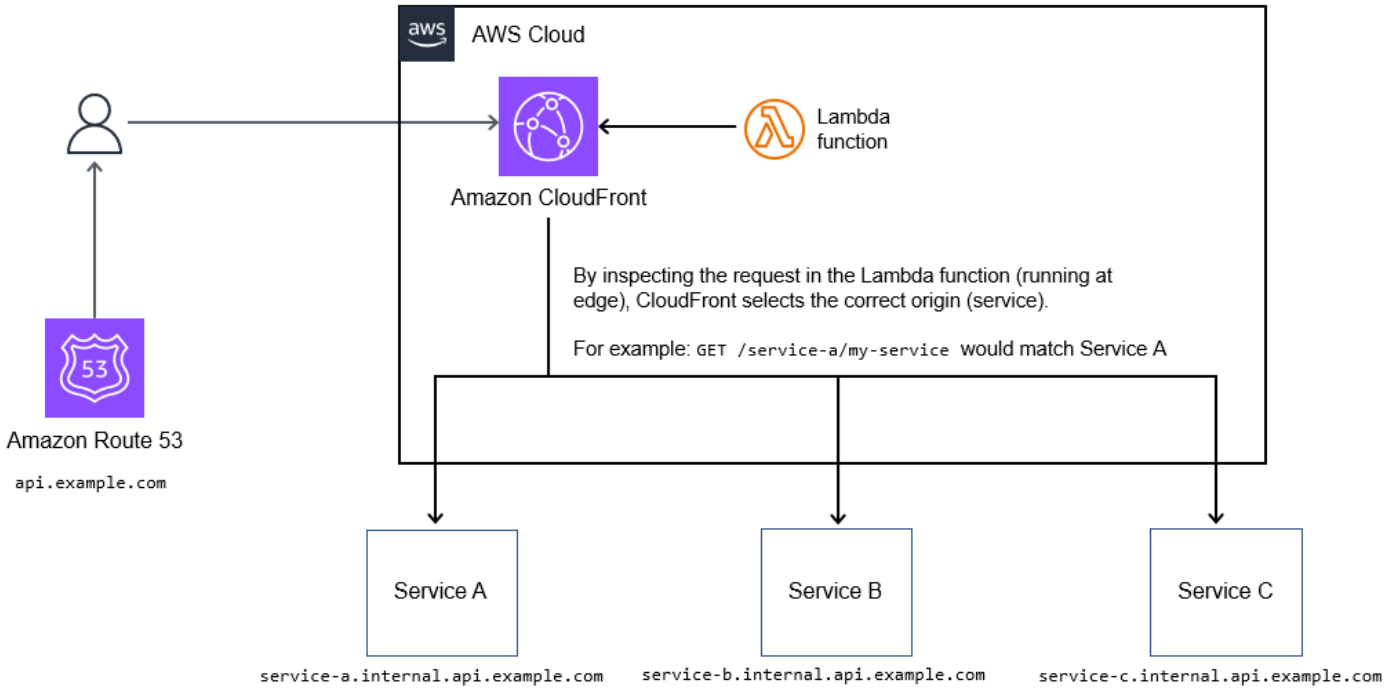
A volumi elevati, il costo potrebbe essere un problema per alcuni utenti.

CloudFront

Puoi utilizzare la funzione di selezione dinamica dell'origine di Amazon CloudFront per selezionare in modo condizionale un'origine (un servizio) per inoltrare la richiesta. Puoi utilizzare questa funzionalità per indirizzare una serie di servizi attraverso un singolo nome host, ad esempio `api.example.com`.

Caso d'uso tipico

La logica di routing risiede come codice all'interno della funzione Lambda @Edge, quindi supporta meccanismi di routing altamente personalizzabili come test A/B, versioni canary, contrassegno delle funzionalità e riscrittura dei percorsi. Il diagramma seguente ne è l'illustrazione.



Pro

Se hai bisogno di memorizzare nella cache le risposte delle API, questo metodo è un ottimo modo per unificare una raccolta di servizi dietro un unico endpoint. È un metodo conveniente per unificare le raccolte di API.

Inoltre, CloudFront supporta la crittografia a livello di campo e l'integrazione con la limitazione della velocità di base e AWS WAF gli ACL di base.

Contro

Questo metodo supporta un massimo di 250 origini (servizi) che possono essere unificate. Questo limite è sufficiente per la maggior parte delle implementazioni, ma potrebbe causare problemi con un gran numero di API man mano che si amplia il portafoglio di servizi.

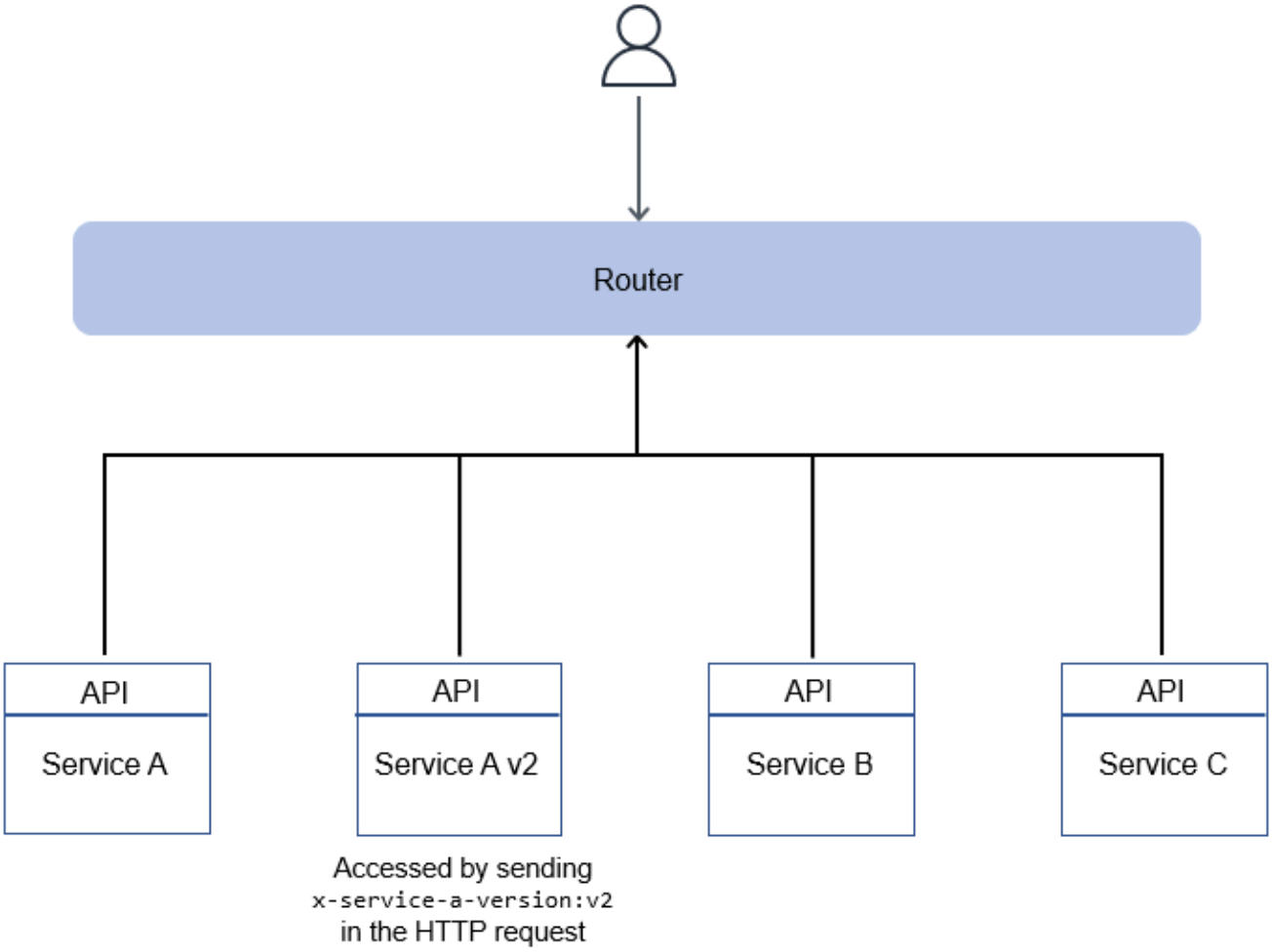
L'aggiornamento delle funzioni Lambda @Edge attualmente richiede alcuni minuti. CloudFront inoltre, richiede fino a 30 minuti per completare la propagazione delle modifiche a tutti i punti di presenza. Questo alla fine blocca ulteriori aggiornamenti fino al loro completamento.

Modello di routing delle intestazioni HTTP

Il routing basato su intestazioni consente di indirizzare il servizio corretto per ogni richiesta specificando un'intestazione HTTP nella richiesta HTTP. Ad esempio, l'invio dell'intestazione `x-service-a-action: get-thing` consentirebbe di eseguire `get thing` da `Service A`. Il percorso della richiesta è comunque importante, perché offre indicazioni sulla risorsa su cui stai cercando di lavorare.

Oltre a utilizzare il routing delle intestazioni HTTP per le azioni, puoi utilizzarlo come meccanismo per il routing delle versioni, abilitare i flag di funzionalità, i test A/B o esigenze simili. In realtà, è probabile che utilizzerai il routing delle intestazioni con uno degli altri metodi di routing per creare API affidabili.

L'architettura per il routing delle intestazioni HTTP in genere prevede un sottile livello di routing davanti ai microservizi che indirizza al servizio corretto e restituisce una risposta, come illustrato nel diagramma seguente. Questo livello di routing può coprire tutti i servizi o solo alcuni servizi per consentire un'operazione come il routing basato sulla versione.



Pro

Le modifiche alla configurazione richiedono uno sforzo minimo e possono essere automatizzate facilmente. Questo metodo è inoltre flessibile e supporta modi creativi per esporre solo le operazioni specifiche che si desidera utilizzare in un servizio.

Contro

Analogamente al metodo di routing del nome host, il routing delle intestazioni HTTP presuppone che l'utente abbia il pieno controllo sul client e che sia possibile manipolare intestazioni HTTP personalizzate. I proxy, le reti di distribuzione di contenuti (CDN) e i sistemi di bilanciamento del carico possono limitare la dimensione dell'intestazione. Sebbene sia improbabile che ciò costituisca un problema, potrebbe trattarsi di un problema a seconda del numero di intestazioni e cookie aggiunti.

Schema dell'interruttore automatico

Intento

Lo schema di interruzione del circuito può impedire a un servizio chiamante di ritentare una chiamata a un altro servizio (chiamante) se in precedenza la chiamata ha causato ripetuti timeout o guasti. Lo schema viene utilizzato anche per rilevare quando il servizio chiamante è di nuovo operativo.

Motivazione

Quando più microservizi collaborano per gestire le richieste, uno o più servizi potrebbero non essere disponibili o presentare una latenza elevata. Quando applicazioni complesse utilizzano microservizi, un'interruzione di un microservizio può causare il fallimento dell'applicazione. I microservizi comunicano tramite chiamate di procedura remote e possono verificarsi errori transitori nella connettività di rete, con conseguenti guasti. (Gli errori transitori possono essere gestiti utilizzando il pattern retry with backoff). Durante l'esecuzione sincrona, la sovrapposizione di timeout o errori può causare un'esperienza utente scadente.

Tuttavia, in alcune situazioni, la risoluzione degli errori potrebbe richiedere più tempo, ad esempio quando il servizio chiamante non è attivo o se un conflitto nel database causa dei timeout. In questi casi, se il servizio chiamante riprova le chiamate ripetutamente, tali tentativi potrebbero causare conflitti di rete e il consumo del pool di thread del database. Inoltre, se più utenti riutilizzano l'applicazione ripetutamente, ciò aggraverà il problema e potrebbe causare un peggioramento delle prestazioni dell'intera applicazione.

Lo schema degli interruttori automatici è stato reso popolare da Michael Nygard nel suo libro Release It (Nygard 2018). Questo modello di progettazione può impedire a un servizio chiamante di ritentare una chiamata di servizio che in precedenza aveva causato ripetuti timeout o guasti. È inoltre in grado di rilevare quando il servizio chiamante è di nuovo operativo.

Gli oggetti dell'interruttore funzionano come interruttori elettrici che interrompono automaticamente la corrente quando si verifica un'anomalia nel circuito. Gli interruttori elettrici interrompono o fanno scattare il flusso di corrente in caso di guasto. Allo stesso modo, l'oggetto interruttore si trova tra il chiamante e il servizio chiamante e interviene se il chiamante non è disponibile.

Gli errori del calcolo distribuito sono un insieme di affermazioni fatte da Peter Deutsch e altri di Sun Microsystems. Dicono che i programmatori che sono nuovi alle applicazioni distribuite invariabilmente

formulano false ipotesi. L'affidabilità della rete, le aspettative di latenza zero e le limitazioni della larghezza di banda fanno sì che le applicazioni software siano scritte con una gestione minima degli errori di rete.

Durante un'interruzione della rete, le applicazioni potrebbero attendere una risposta a tempo indeterminato e consumare continuamente le risorse dell'applicazione. La mancata ripetizione delle operazioni quando la rete diventa disponibile può anche portare al degrado delle applicazioni. Se le chiamate API a un database o a un servizio esterno si interrompono a causa di problemi di rete, le chiamate ripetute senza interruttori automatici possono influire sui costi e sulle prestazioni.

Applicabilità

Usa questo schema quando:

- Il servizio chiamante effettua una chiamata che molto probabilmente non andrà a buon fine.
- Un'elevata latenza mostrata dal servizio chiamante (ad esempio, quando le connessioni al database sono lente) causa dei timeout per il servizio chiamante.
- Il servizio chiamante effettua una chiamata sincrona, ma il servizio chiamante non è disponibile o presenta una latenza elevata.

Problemi e considerazioni

- Implementazione indipendente dal servizio: per evitare il gonfiamento del codice, si consiglia di implementare l'oggetto circuit breaker in modo indipendente dai microservizi e basato sull'API.
- Chiusura del circuito da parte del chiamante: quando il chiamante si riprende dal problema o dal guasto delle prestazioni, può aggiornare lo stato del circuito a. `CLOSED` Si tratta di un'estensione dello schema degli interruttori automatici e può essere implementata se il Recovery Time Objective (RTO) lo richiede.
- Chiamate multithread: il valore di timeout di scadenza è definito come il periodo di tempo in cui il circuito rimane attivo prima che le chiamate vengano nuovamente instradate per verificare la disponibilità del servizio. Quando il servizio chiamante viene chiamato in più thread, la prima chiamata non riuscita definisce il valore del timeout di scadenza. L'implementazione deve garantire che le chiamate successive non spostino il timeout di scadenza all'infinito.
- Apertura o chiusura forzata del circuito: gli amministratori di sistema devono avere la possibilità di aprire o chiudere un circuito. Ciò può essere fatto aggiornando il valore del timeout di scadenza nella tabella del database.

- Osservabilità: l'applicazione deve avere una registrazione configurata per identificare le chiamate che falliscono quando l'interruttore è aperto.

Implementazione

Architettura di alto livello

Nell'esempio seguente, il chiamante è il servizio ordini e il chiamante è il servizio di pagamento.

In assenza di guasti, il servizio ordini indirizza tutte le chiamate al servizio di pagamento tramite l'interruttore automatico, come illustrato nello schema seguente.

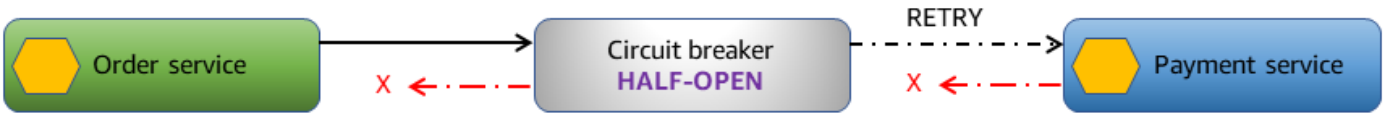
In caso di timeout del servizio di pagamento, l'interruttore automatico può rilevare il timeout e tracciare l'errore.

Circuit breaker with payment service failure

Se i timeout superano una soglia specificata, l'applicazione apre il circuito. Quando il circuito è aperto, l'oggetto interruttore automatico non indirizza le chiamate al servizio di pagamento. Quando il servizio di ordinazione chiama il servizio di pagamento, restituisce un errore immediato.

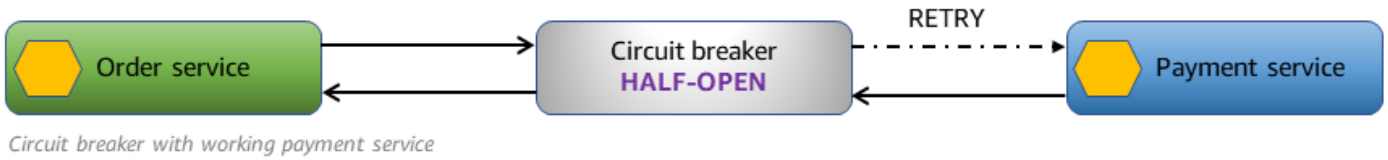
Circuit breaker stops routing to payment service

L'oggetto circuit breaker cerca periodicamente di verificare se le chiamate al servizio di pagamento hanno esito positivo.



Circuit breaker periodically retries payment service

Quando la chiamata al servizio di pagamento ha esito positivo, il circuito viene chiuso e tutte le altre chiamate vengono nuovamente indirizzate al servizio di pagamento.



Circuit breaker with working payment service

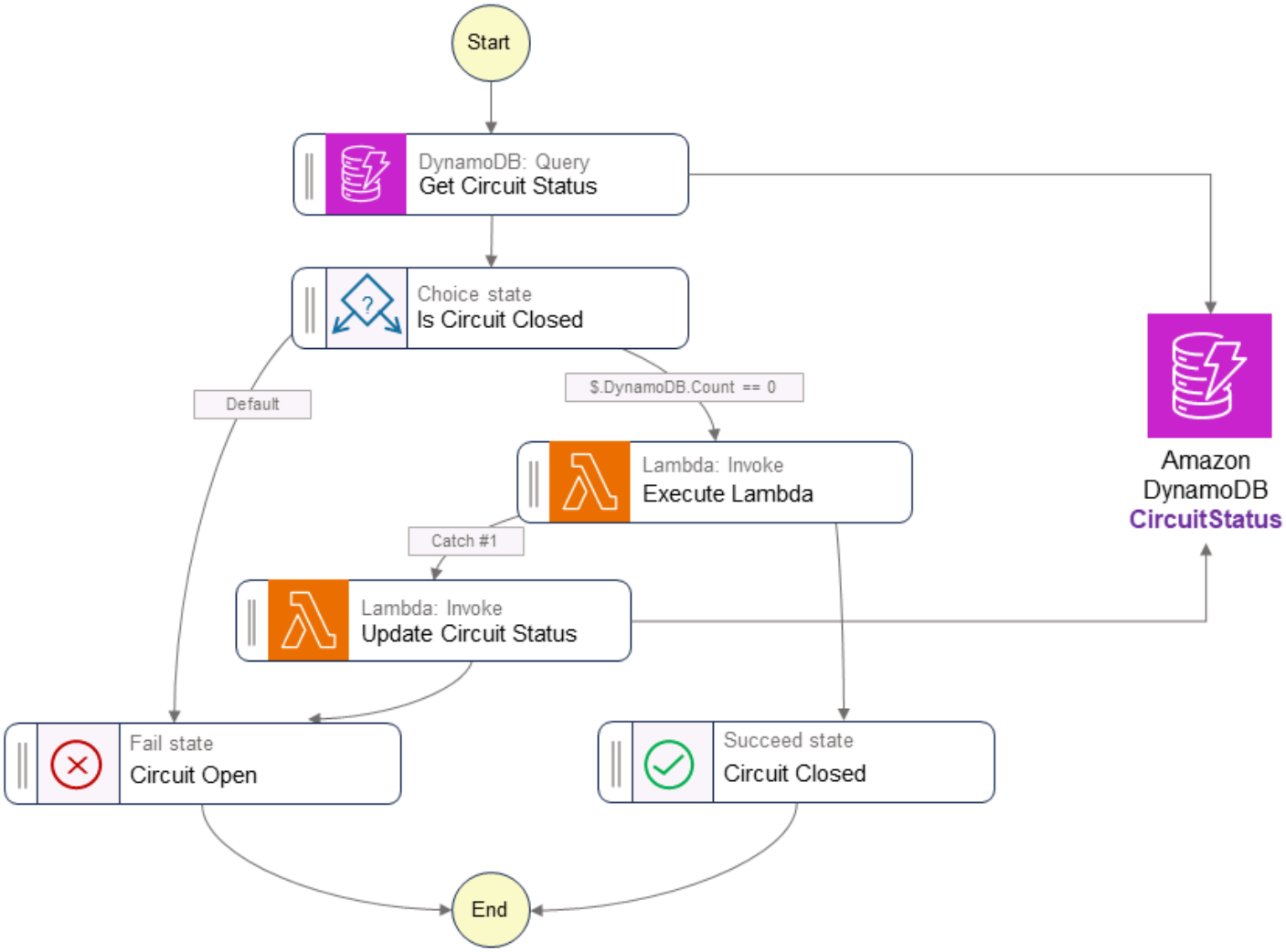
Implementazione tramite servizi AWS

La soluzione di esempio utilizza flussi di lavoro rapidi AWS Step Functionsper implementare lo schema degli interruttori automatici. La macchina a stati Step Functions consente di configurare le funzionalità di riprova e il flusso di controllo basato sulle decisioni necessari per l'implementazione del modello.

La soluzione utilizza anche una tabella Amazon DynamoDB come archivio dati per tracciare lo stato del circuito. Questo può essere sostituito con un datastore in memoria come Amazon ElastiCache for Redis per prestazioni migliori.

Quando un servizio desidera chiamare un altro servizio, avvia il flusso di lavoro con il nome del servizio chiamante. Il flusso di lavoro ottiene lo stato dell'interruttore automatico dalla tabella `CircuitStatus` DynamoDB, che memorizza i servizi attualmente danneggiati. Se `CircuitStatus` contiene un record non scaduto per il chiamante, il circuito è aperto. Il flusso di lavoro Step Functions restituisce un errore immediato e termina con uno `FAIL` stato.

Se la `CircuitStatus` tabella non contiene un record per il chiamante o contiene un record scaduto, il servizio è operativo. Il `ExecuteLambda` passaggio nella definizione della macchina a stati richiama la funzione Lambda che viene inviata tramite un valore di parametro. Se la chiamata ha esito positivo, il flusso di lavoro Step Functions termina con uno `SUCCESS` stato.



Se la chiamata di servizio fallisce o si verifica un timeout, l'applicazione riprova con un backoff esponenziale per un numero definito di volte. Se la chiamata di servizio fallisce dopo i nuovi tentativi, il flusso di lavoro inserisce un record nella `CircuitStatus` tabella del servizio con l'an `ExpiryTimeStamp` e il flusso di lavoro termina con uno stato. `FAIL` Le chiamate successive allo stesso servizio restituiscono un errore immediato finché l'interruttore è aperto. La `Get Circuit Status` fase di definizione della macchina a stati verifica la disponibilità del servizio in base al `ExpiryTimeStamp` valore. Gli elementi scaduti vengono eliminati dalla `CircuitStatus` tabella utilizzando la funzionalità time to live (TTL) di DynamoDB.

Codice di esempio

Il codice seguente utilizza la funzione `GetCircuitStatus` Lambda per controllare lo stato dell'interruttore.

```
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 = "";
}
```

Il codice seguente mostra le istruzioni di Amazon States Language nel flusso di lavoro 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 deposito

Per un'implementazione completa dell'architettura di esempio per questo pattern, consultate il GitHub repository all'indirizzo https://github.com/aws-samples/. circuit-breaker-netcore-blog

Riferimenti del blog

- Utilizzo del pattern di interruttore automatico con AWS Step Functions e Amazon DynamoDB

Contenuti correlati

- Schema Strangler fig
- Nuovo tentativo con schema di backoff
- AWS App Meshfunzionalità di interruttore automatico

Modello di origine evento

Intento

Nelle architetture basate sugli eventi, il modello di approvvigionamento degli eventi memorizza gli eventi che determinano un cambiamento di stato in un datastore. Ciò consente di acquisire e mantenere una cronologia completa dei cambiamenti di stato e promuove la verificabilità, la tracciabilità e la capacità di analizzare gli stati passati.

Motivazione

Più microservizi possono collaborare per gestire le richieste e comunicare attraverso eventi. Questi eventi possono comportare un cambiamento di stato (dati). La memorizzazione degli oggetti evento nell'ordine in cui si presentano fornisce informazioni preziose sullo stato corrente dell'entità dati e informazioni aggiuntive su come è arrivata a tale stato.

Applicabilità

Utilizza il modello di approvvigionamento degli eventi quando:

- Per il tracciamento è necessaria una cronologia immutabile degli eventi che si verificano in un'applicazione.
- Le proiezioni di dati poliglotti sono necessarie da un'unica fonte di verità (SSOT).
- È necessaria la ricostruzione temporale dello stato dell'applicazione.
- L'archiviazione a lungo termine dello stato dell'applicazione non è richiesta, ma è possibile ricostruirla secondo necessità.
- I carichi di lavoro hanno volumi di lettura e scrittura diversi. Ad esempio, hai carichi di lavoro ad alta intensità di scrittura che non richiedono l'elaborazione in tempo reale.
- È necessaria l'acquisizione dei dati di modifica per analizzare le prestazioni dell'applicazione e altri parametri.
- Sono necessari i dati di audit per tutti gli eventi che si verificano in un sistema a fini di reporting e conformità.
- Si desidera ricavare scenari ipotetici modificando (inserendo, aggiornando o eliminando) gli eventi durante il processo di riproduzione per determinare il possibile stato finale.

Problemi e considerazioni

- Controllo ottimistico della concorrenza: questo modello memorizza ogni evento che causa un cambiamento di stato nel sistema. Più utenti o servizi possono provare ad aggiornare lo stesso dato contemporaneamente, causando collisioni di eventi. Queste collisioni si verificano quando vengono creati e applicati contemporaneamente eventi in conflitto, il che si traduce in uno stato finale dei dati che non corrisponde alla realtà. Per risolvere questo problema, puoi implementare strategie per rilevare e risolvere le collisioni di eventi. Ad esempio, è possibile implementare uno schema ottimistico di controllo della concorrenza includendo il controllo delle versioni o aggiungendo timestamp agli eventi per tenere traccia dell'ordine degli aggiornamenti.
- Complessità: l'implementazione dell'approvvigionamento degli eventi richiede un cambiamento di mentalità dalle tradizionali operazioni CRUD a un modo di pensare basato sugli eventi. Il processo di riproduzione, utilizzato per ripristinare il sistema allo stato originale, può essere complesso per garantire l'idempotenza dei dati. L'archiviazione degli eventi, i backup e gli snapshot possono inoltre aggiungere ulteriore complessità.
- Coerenza finale: le proiezioni dei dati derivate dagli eventi sono in ultima analisi coerenti a causa della latenza nell'aggiornamento dei dati mediante il modello CQRS (Command Query Responsibility Segregation) o le viste materializzate. Quando i consumer elaborano i dati da un archivio eventi e gli editori inviano nuovi dati, la proiezione dei dati o l'oggetto dell'applicazione potrebbero non rappresentare lo stato corrente.
- Interrogazione: il recupero di dati correnti o aggregati dai log degli eventi può essere più complesso e più lento rispetto ai database tradizionali, in particolare per query e attività di reporting complesse. Per mitigare questo problema, l'approvvigionamento degli eventi viene spesso implementato con il modello CQRS.
- Dimensioni e costo dell'archivio eventi: l'archivio eventi può registrare una crescita esponenziale delle dimensioni man mano che gli eventi persistono continuamente, specialmente nei sistemi che hanno un numero elevato di eventi o periodi di conservazione prolungati. Di conseguenza, è necessario archiviare periodicamente i dati degli eventi in uno spazio di archiviazione conveniente per evitare che l'archivio eventi diventi troppo grande.
- Scalabilità dell'archivio eventi: l'archivio eventi deve gestire in modo efficiente volumi elevati di operazioni di scrittura e lettura. La scalabilità di un archivio eventi può essere difficile, quindi è importante disporre di un datastore che fornisca frammenti e partizioni.
- Efficienza e ottimizzazione: scegli o progetta un archivio eventi che gestisca le operazioni di scrittura e lettura in modo efficiente. L'archivio eventi deve essere ottimizzato per il volume di eventi e i modelli di query previsti per l'applicazione. L'implementazione di meccanismi di indicizzazione

e interrogazione può accelerare il recupero degli eventi durante la ricostruzione dello stato dell'applicazione. È inoltre possibile prendere in considerazione l'utilizzo di database o librerie specializzati in archivi di eventi che offrono funzionalità di ottimizzazione delle query.

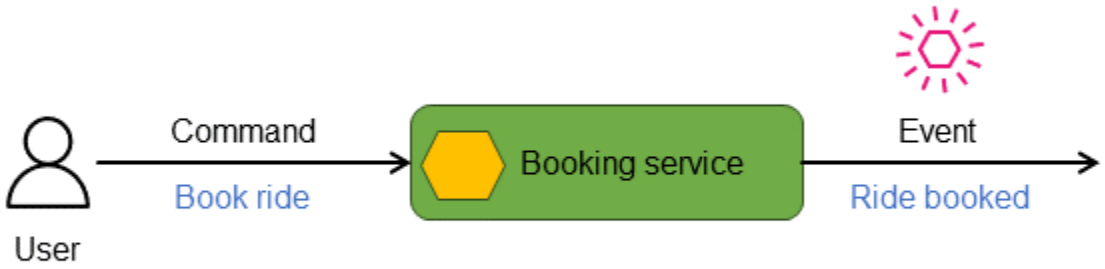
- Snapshot: è necessario eseguire il backup dei log degli eventi a intervalli regolari con attivazione basata sul tempo. La riproduzione degli eventi sull'ultimo backup dei dati riuscito noto dovrebbe portare al ripristino puntuale dello stato dell'applicazione. L'obiettivo del punto di ripristino (RPO) è il periodo di tempo massimo accettabile dall'ultimo punto di ripristino dei dati. L'RPO determina ciò che viene considerato una perdita di dati accettabile tra l'ultimo punto di ripristino e l'interruzione del servizio. La frequenza degli snapshot giornalieri del datastore e dell'archivio eventi deve essere basata sull'RPO dell'applicazione.
- Sensibilità temporale: gli eventi vengono archiviati nell'ordine in cui si verificano. Pertanto, quando si implementa questo modello l'affidabilità della rete è un fattore importante da considerare. I problemi di latenza possono portare a uno stato errato del sistema. Utilizza le code FIFO (first in, first out) con consegna al massimo una volta per portare gli eventi all'archivio eventi.
- Prestazioni della riproduzione degli eventi: la riproduzione di un numero considerevole di eventi per ricostruire lo stato corrente dell'applicazione può richiedere molto tempo. Sono necessari sforzi di ottimizzazione per migliorare le prestazioni, in particolare quando si riproducono gli eventi dai dati archiviati.
- Aggiornamenti di sistema esterni: le applicazioni che utilizzano il modello di approvvigionamento degli eventi potrebbero aggiornare i datastore in sistemi esterni e acquisire questi aggiornamenti come oggetti evento. Durante le riproduzioni degli eventi, questo potrebbe diventare un problema se il sistema esterno non prevede un aggiornamento. In questi casi, è possibile utilizzare i flag di funzionalità per controllare gli aggiornamenti di sistema esterni.
- Interrogazioni di sistemi esterni: quando le chiamate ai sistemi esterni sono sensibili alla data e all'ora della chiamata, i dati ricevuti possono essere archiviati in datastore interni per essere utilizzati durante le riproduzioni.
- Controllo delle versioni degli eventi: man mano che l'applicazione si evolve, la struttura degli eventi (modello) può cambiare. È quindi necessaria l'implementazione di una strategia di controllo delle versioni degli eventi per garantire la compatibilità con le versioni precedenti e successive. Ciò può comportare l'inclusione di un campo di versione nel payload dell'evento e la gestione appropriata delle diverse versioni dell'evento durante la riproduzione.

Implementazione

Architettura di alto livello

Comandi ed eventi

Nelle applicazioni di microservizi distribuite e basate sugli eventi, i comandi rappresentano le istruzioni o le richieste inviate a un servizio, in genere con l'intento di avviare una modifica del suo stato. Il servizio elabora questi comandi e valuta la validità e l'applicabilità del comando allo stato corrente. Se il comando viene eseguito correttamente, il servizio risponde emettendo un evento che indica l'azione intrapresa e le informazioni pertinenti sullo stato. Ad esempio, nel diagramma seguente, il servizio di prenotazione risponde al comando Book ride (Prenota corsa) emettendo l'evento Ride booked (Corsa prenotata).



Archivi di eventi

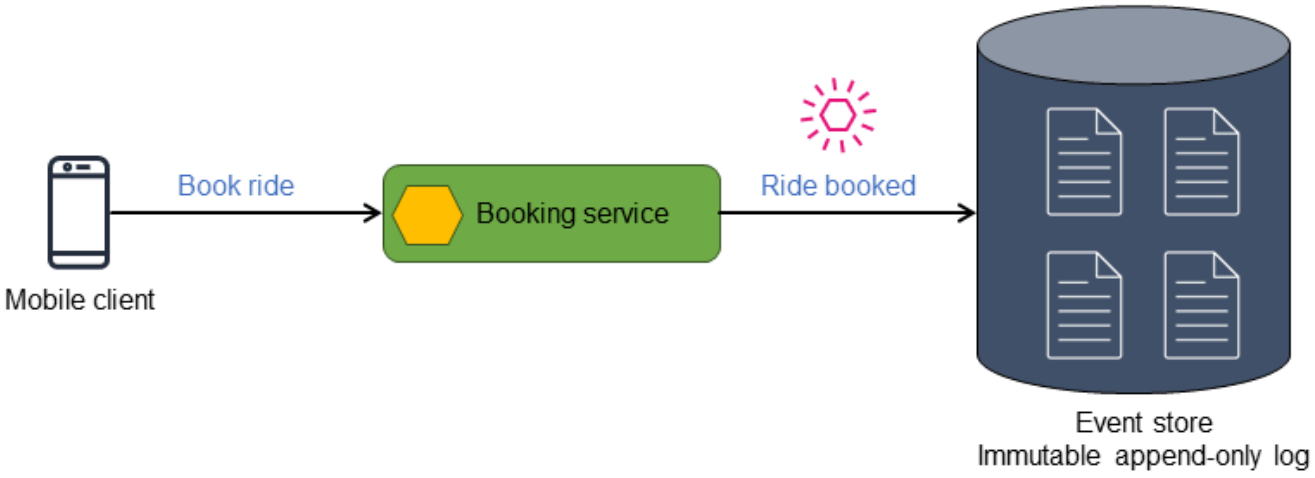
Gli eventi vengono registrati in un repository o datastore immutabile, di sola aggiunta e ordinato cronologicamente, noto come archivio eventi Ogni modifica di stato viene trattata come un singolo oggetto evento. È possibile ricostruire un oggetto di entità o un datastore con uno stato iniziale noto, lo stato corrente e qualsiasi visualizzazione temporale ripetendo gli eventi nell'ordine in cui si sono verificati.

L'archivio eventi funge da registrazione storica di tutte le azioni e i cambiamenti di stato e funge da preziosa fonte di informazioni. È possibile utilizzare l'archivio eventi per derivare lo stato finale e aggiornato del sistema passando gli eventi attraverso un processore di riproduzione, che applica questi eventi per produrre una rappresentazione accurata dello stato più recente del sistema. È inoltre possibile utilizzare l'archivio eventi per generare la prospettiva puntuale dello stato ripetendo gli eventi tramite un processore di riproduzione. Nel modello di approvvigionamento degli eventi, lo

stato corrente potrebbe non essere interamente rappresentato dall'oggetto evento più recente. Puoi derivare lo stato corrente in uno dei seguenti tre modi:

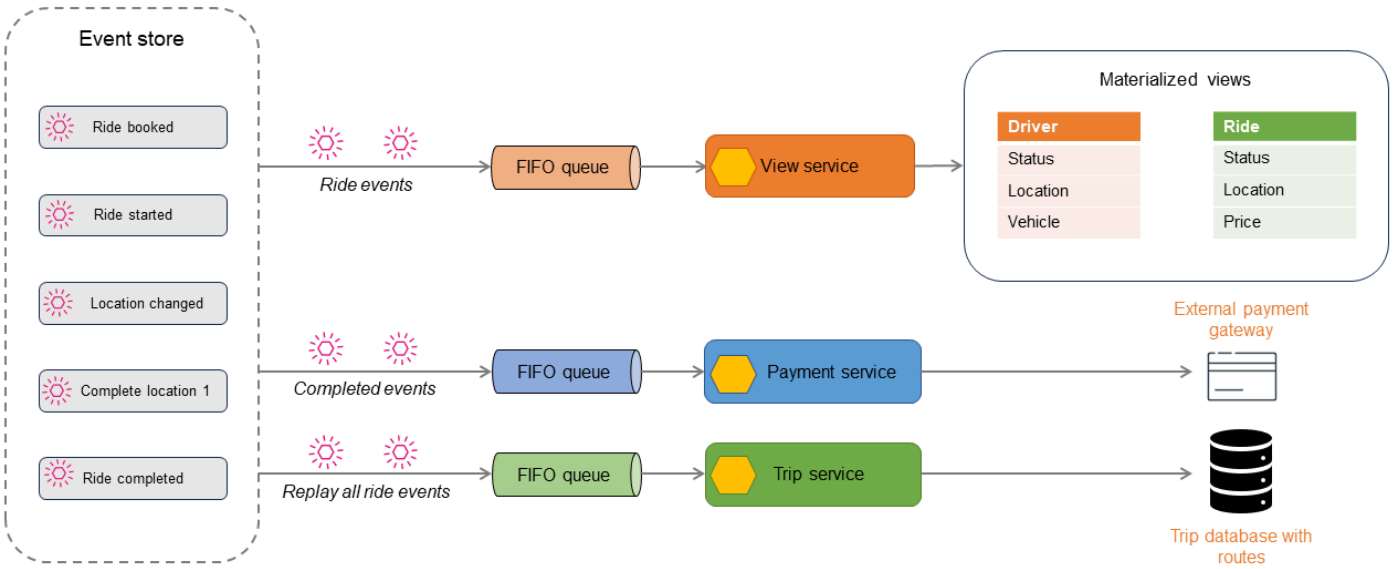
- Aggregando gli eventi correlati. Gli oggetti evento correlati vengono combinati per generare lo stato corrente per l'interrogazione. Questo approccio viene spesso utilizzato insieme al modello CQRS, in quanto gli eventi vengono combinati e scritti nel datastore di sola lettura.
- Utilizzando le viste materializzate. È possibile utilizzare l'approvvigionamento degli eventi con il modello della vista materializzata per calcolare o riepilogare i dati degli eventi e ottenere lo stato corrente dei dati correlati.
- Riproducendo gli eventi. Gli oggetti evento possono essere riprodotti per eseguire azioni volte a generare lo stato corrente.

Il diagramma seguente mostra l'evento `Ride booked` archiviato in un archivio eventi.



L'archivio eventi pubblica gli eventi che memorizza e gli eventi possono essere filtrati e indirizzati al processore appropriato per le azioni successive. Ad esempio, gli eventi possono essere indirizzati a un processore di visualizzazione che riepiloga lo stato e mostra una vista materializzata. Gli eventi vengono trasformati nel formato di dati del datastore di destinazione. Questa architettura può essere estesa per derivare diversi tipi di datastore, il che porta alla persistenza poliglotta dei dati.

Il diagramma seguente descrive gli eventi in un'applicazione di prenotazione di una corsa auto. Tutti gli eventi che si verificano all'interno dell'applicazione vengono archiviati nell'archivio eventi. Gli eventi memorizzati vengono quindi filtrati e indirizzati a diversi consumer.



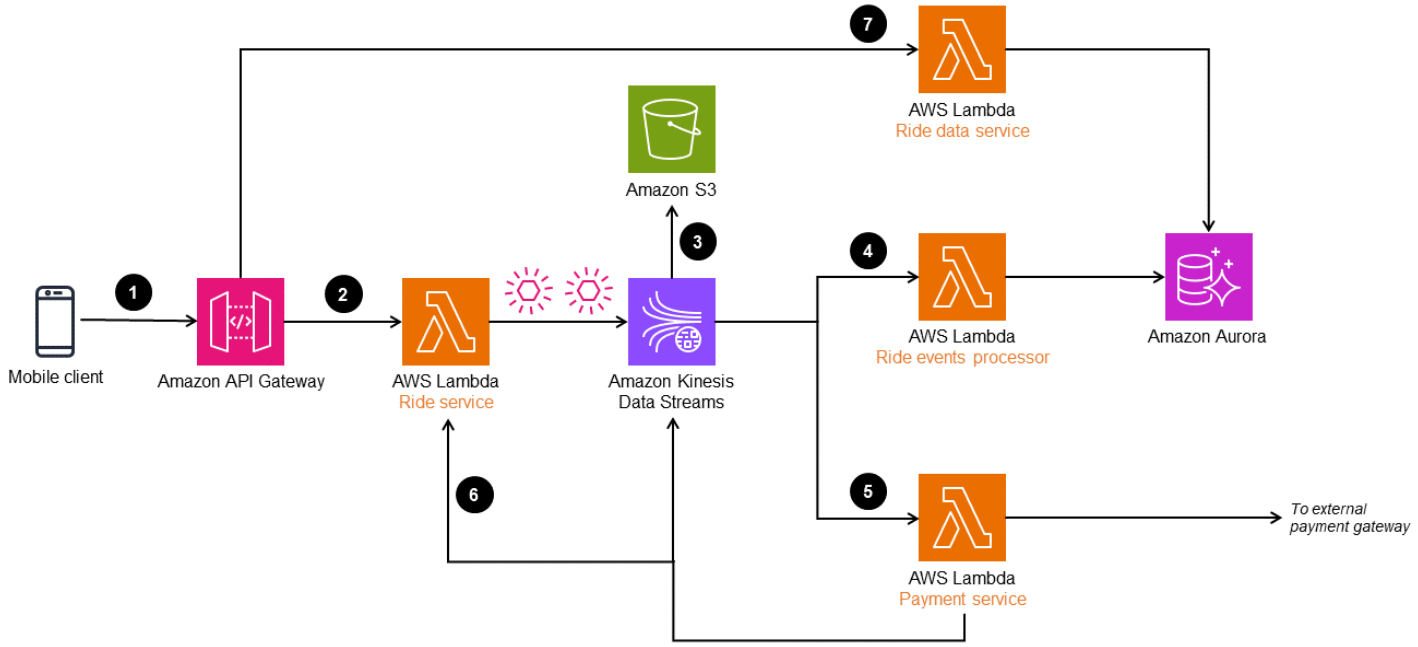
Gli eventi della corsa possono essere utilizzati per generare datastore di sola lettura utilizzando il CQRS o il modello della vista materializzata. Puoi ottenere lo stato attuale della corsa, il conducente o la prenotazione interrogando gli archivi di lettura. Alcuni eventi, come `Location changed` o `Ride completed`, vengono pubblicati su un altro consumer per l'elaborazione dei pagamenti. Una volta completata la corsa, tutti gli eventi della corsa vengono riprodotti per creare una cronologia della corsa a scopo di controllo o rendicontazione.

Il modello di approvvigionamento degli eventi viene spesso utilizzato in applicazioni che richiedono un ripristino puntuale e anche quando i dati devono essere proiettati in formati diversi utilizzando un'unica fonte di verità. Entrambe queste operazioni richiedono un processo di ripetizione per eseguire gli eventi e determinare lo stato finale richiesto. Il processore di riproduzione potrebbe inoltre richiedere un punto di partenza noto, idealmente non all'avvio dell'applicazione, perché non sarebbe un processo efficiente. Si consiglia di acquisire snapshot regolari dello stato del sistema e di applicare un numero inferiore di eventi per ricavare uno stato aggiornato.

Implementazione tramite servizi AWS

Nella seguente architettura, il flusso di dati Amazon Kinesis viene utilizzato come archivio eventi. Questo servizio acquisisce e gestisce le modifiche delle applicazioni sotto forma di eventi e offre una soluzione di streaming di dati ad alta velocità di trasmissione effettiva e in tempo reale. Per implementare il modello di approvvigionamento degli eventi su AWS, puoi anche utilizzare servizi come Amazon EventBridge e Streaming gestito da Amazon per Apache Kafka (Amazon MSK) in base alle esigenze della tua applicazione.

Per migliorare la durabilità e abilitare il controllo, puoi archiviare gli eventi acquisiti dal flusso di dati Kinesis in Amazon Simple Storage Service (Amazon S3). Questo approccio a doppia archiviazione aiuta a conservare i dati storici degli eventi in modo sicuro per future analisi e scopi di conformità.



Il flusso di lavoro consiste nei seguenti passaggi:

1. Una richiesta di prenotazione della corsa viene effettuata tramite un client mobile verso un endpoint Gateway Amazon API.
2. Il microservizio della corsa (funzione `Ride service` Lambda) riceve la richiesta, trasforma gli oggetti e li pubblica sul flusso di dati Kinesis.
3. I dati degli eventi nel flusso di dati Kinesis vengono archiviati in Amazon S3 a scopo di conformità e cronologia degli audit.
4. Gli eventi vengono trasformati ed elaborati dalla funzione `Ride event processor` Lambda e archiviati in un database Amazon Aurora per fornire una vista materializzata dei dati della corsa.
5. Gli eventi della corsa completati vengono filtrati e inviati per l'elaborazione dei pagamenti a un gateway di pagamento esterno. Una volta completato il pagamento, viene inviato un altro evento al flusso di dati Kinesis per aggiornare il database Ride (Corse).
6. Al termine della corsa, gli eventi di viaggio vengono riprodotti nella funzione `Ride service` Lambda per creare i percorsi e la cronologia della corsa.
7. Le informazioni sulla corsa possono essere lette tramite il `Ride data service`, che legge dal database Aurora.

Gateway API può anche inviare l'oggetto evento direttamente al flusso di dati Kinesis senza la funzione `Ride service` Lambda. Tuttavia, in un sistema complesso come un servizio di ride hailing, potrebbe essere necessario elaborare e arricchire l'oggetto evento prima di inserirlo nel flusso di dati. Per questo motivo, l'architettura dispone di un `Ride service` che elabora l'evento prima di inviarlo al flusso di dati Kinesis.

Riferimenti del blog

- Novità per AWS Lambda: FIFO SQS come origine evento

Modello di architettura esagonale

Intento

Il modello di architettura esagonale, noto anche come pattern di porte e adattatori, è stato proposto dal Dr. Alistair Cockburn nel 2005. Mira a creare architetture liberamente accoppiate in cui i componenti delle applicazioni possano essere testati in modo indipendente, senza dipendenze da archivi di dati o interfacce utente (UI). Questo modello aiuta a prevenire il blocco tecnologico degli archivi di dati e delle interfacce utente. Ciò semplifica la modifica dello stack tecnologico nel tempo, con un impatto limitato o nullo sulla logica aziendale. In questa architettura ad accoppiamento libero, l'applicazione comunica con componenti esterni tramite interfacce chiamate porte e utilizza adattatori per tradurre gli scambi tecnici con questi componenti.

Motivazione

Il modello di architettura esagonale viene utilizzato per isolare la logica aziendale (logica di dominio) dal codice dell'infrastruttura correlato, ad esempio il codice per accedere a un database o alle API esterne. Questo modello è utile per creare logica aziendale e codice di infrastruttura liberamente accoppiati per AWS Lambda funzioni che richiedono l'integrazione con servizi esterni. Nelle architetture tradizionali, una pratica comune consiste nell'incorporare la logica di business nel livello del database come procedure archiviate e nell'interfaccia utente. Questa pratica, oltre all'utilizzo di costrutti specifici dell'interfaccia utente all'interno della logica aziendale, porta alla creazione di architetture strettamente collegate che causano rallentamenti nelle migrazioni dei database e negli sforzi di modernizzazione dell'esperienza utente (UX). Il modello di architettura esagonale consente di progettare i sistemi e le applicazioni in base allo scopo anziché in base alla tecnologia. Questa strategia si traduce in componenti applicativi facilmente intercambiabili come database, UX e componenti di servizio.

Applicabilità

Utilizzate il modello di architettura esagonale quando:

- Desiderate disaccoppiare l'architettura dell'applicazione per creare componenti che possano essere completamente testati.
- Più tipi di client possono utilizzare la stessa logica di dominio.

- I componenti dell'interfaccia utente e del database richiedono aggiornamenti tecnologici periodici che non influiscono sulla logica dell'applicazione.
- L'applicazione richiede più fornitori di input e consumatori di output e la personalizzazione della logica dell'applicazione comporta complessità del codice e mancanza di estensibilità.

Problemi e considerazioni

- Progettazione basata sul dominio: l'architettura esagonale funziona particolarmente bene con la progettazione basata sul dominio (DDD). Ogni componente dell'applicazione rappresenta un sottodominio in DDD e le architetture esagonali possono essere utilizzate per ottenere un accoppiamento libero tra i componenti dell'applicazione.
- Testabilità: in base alla progettazione, un'architettura esagonale utilizza astrazioni per input e output. Pertanto, scrivere test unitari e test in modo isolato diventa più semplice grazie all'accoppiamento libero intrinseco.
- Complessità: la complessità della separazione della logica aziendale dal codice dell'infrastruttura, se gestita con attenzione, può offrire grandi vantaggi come agilità, copertura dei test e adattabilità tecnologica. In caso contrario, i problemi possono diventare complessi da risolvere.
- Sovraccarico di manutenzione: il codice adattatore aggiuntivo che rende l'architettura collegabile è giustificato solo se il componente dell'applicazione richiede diverse sorgenti di input e destinazioni di output su cui scrivere, o quando l'archivio dati di input e output deve cambiare nel tempo. In caso contrario, l'adattatore diventa un altro livello aggiuntivo da gestire, il che comporta un sovraccarico di manutenzione.
- Problemi di latenza: l'uso di porte e adattatori aggiunge un altro livello, che potrebbe causare latenza.

Implementazione

Le architetture esagonali supportano l'isolamento della logica applicativa e aziendale dal codice dell'infrastruttura e dal codice che integra l'applicazione con interfacce utente, API esterne, database e broker di messaggi. È possibile collegare facilmente i componenti della logica aziendale ad altri componenti (come i database) nell'architettura dell'applicazione tramite porte e adattatori.

Le porte sono punti di ingresso indipendenti dalla tecnologia in un componente dell'applicazione. Queste interfacce personalizzate determinano l'interfaccia che consente agli attori esterni di

comunicare con il componente dell'applicazione, indipendentemente da chi o cosa implementa l'interfaccia. È simile al modo in cui una porta USB consente a molti tipi diversi di dispositivi di comunicare con un computer, purché utilizzino un adattatore USB.

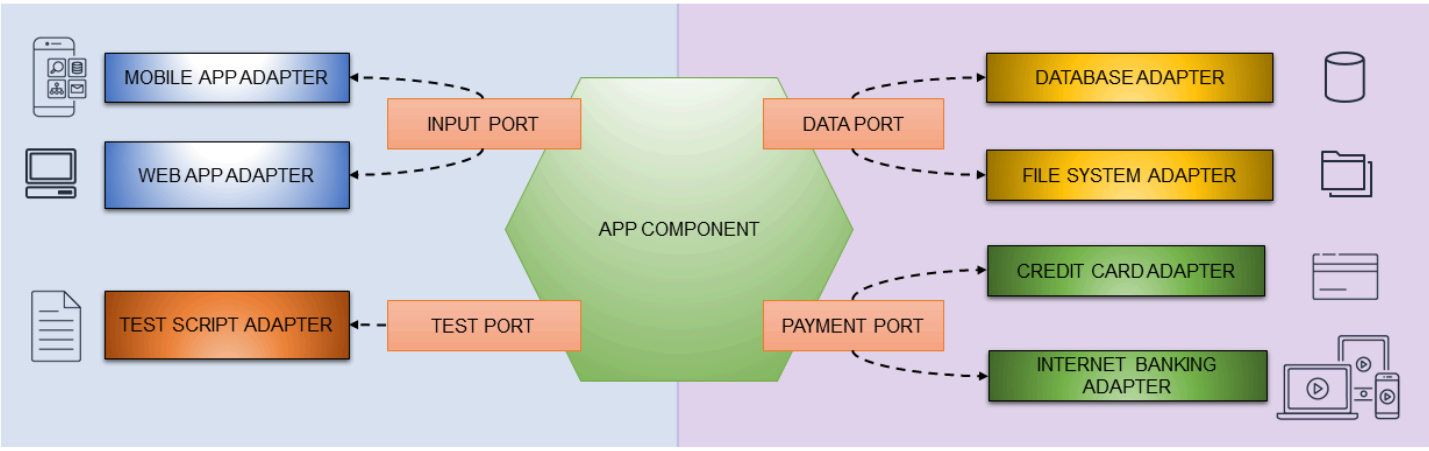
Gli adattatori interagiscono con l'applicazione tramite una porta utilizzando una tecnologia specifica. Gli adattatori si collegano a queste porte, ricevono o forniscono dati alle porte e li trasformano per un'ulteriore elaborazione. Ad esempio, un adattatore REST consente agli attori di comunicare con il componente dell'applicazione tramite un'API REST. Una porta può avere più adattatori senza alcun rischio per la porta o il componente dell'applicazione. Per estendere l'esempio precedente, l'aggiunta di un adattatore GraphQL alla stessa porta fornisce agli attori un mezzo aggiuntivo per interagire con l'applicazione tramite un'API GraphQL senza influire sull'API REST, sulla porta o sull'applicazione.

Le porte si connettono all'applicazione e gli adattatori fungono da connessione con il mondo esterno. È possibile utilizzare le porte per creare componenti applicativi liberamente accoppiati e scambiare componenti dipendenti cambiando l'adattatore. Ciò consente al componente dell'applicazione di interagire con input e output esterni senza la necessità di avere alcuna conoscenza del contesto. I componenti sono intercambiabili a qualsiasi livello, il che facilita i test automatici. È possibile testare i componenti in modo indipendente senza dipendere dal codice dell'infrastruttura anziché fornire un intero ambiente per eseguire i test. La logica dell'applicazione non dipende da fattori esterni, quindi i test sono semplificati e diventa più facile simulare le dipendenze.

Ad esempio, in un'architettura scarsamente accoppiata, un componente dell'applicazione dovrebbe essere in grado di leggere e scrivere dati senza conoscere i dettagli del data store. La responsabilità del componente applicativo è fornire dati a un'interfaccia (porta). Un adattatore definisce la logica di scrittura su un data store, che può essere un database, un file system o un sistema di storage di oggetti come Amazon S3, a seconda delle esigenze dell'applicazione.

Architettura di alto livello

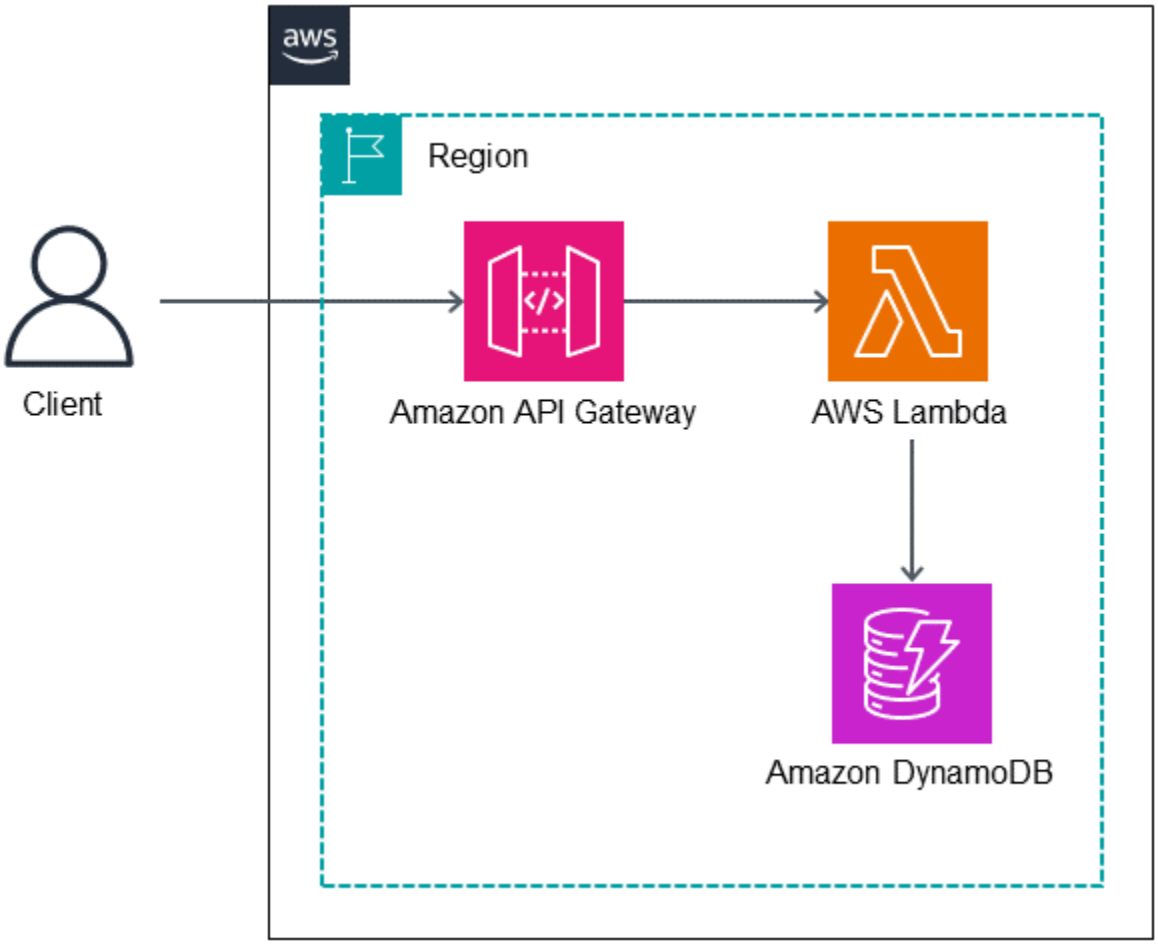
L'applicazione o il componente dell'applicazione contiene la logica aziendale principale. Riceve comandi o interrogazioni dalle porte e invia le richieste tramite le porte ad attori esterni, che vengono implementate tramite adattatori, come illustrato nel diagramma seguente.



Implementazione utilizzando Servizi AWS

AWS Lambda le funzioni spesso contengono sia la logica aziendale che il codice di integrazione del database, che sono strettamente associati per raggiungere un obiettivo. È possibile utilizzare il modello di architettura esagonale per separare la logica aziendale dal codice dell'infrastruttura. Questa separazione consente il test unitario della logica di business senza alcuna dipendenza dal codice del database e migliora l'agilità del processo di sviluppo.

Nella seguente architettura, una funzione Lambda implementa il modello di architettura esagonale. La funzione Lambda viene avviata dall'API REST di Amazon API Gateway. La funzione implementa la logica aziendale e scrive dati nelle tabelle DynamoDB.



Codice di esempio

Il codice di esempio in questa sezione mostra come implementare il modello di dominio utilizzando Lambda, separarlo dal codice dell'infrastruttura (ad esempio il codice per accedere a DynamoDB) e implementare il test unitario per la funzione.

Modello di dominio

La classe del modello di dominio non conosce componenti o dipendenze esterne, ma implementa solo la logica di business. Nell'esempio seguente, la classe `Recipient` è una classe del modello di dominio che verifica le sovrapposizioni nella data di prenotazione.

```
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:
.....
```

Porta di ingresso

La `RecipientInputPort` classe si connette alla classe destinataria ed esegue la logica del dominio.

```
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 di adattatori DynamoDB

La `DDBRecipientAdapter` classe implementa l'accesso alle tabelle 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 funzione Lambda `get_recipient_input_port` è una fabbrica per le istanze della classe. `RecipientInputPort` Costruisce istanze di classi di porte di output con istanze di adattatore correlate.

```
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
}),
}
```

Test unitari

È possibile testare la logica di business per le classi del modello di dominio iniettando classi fittizie. L'esempio seguente fornisce il test unitario per la classe del modello `Recipent` di dominio.

```
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 repository

Per un'implementazione completa dell'architettura di esempio per questo modello, consulta il GitHub repository all'indirizzo https://github.com/aws-samples/ aws-lambda-domain-model -sample.

Contenuti correlati

- Architettura esagonale, articolo di Alistair Cockburn
- Sviluppo di architetture evolutive con (post sul blog in giapponese) AWS LambdaAWS

Video

Il video seguente (in giapponese) illustra l'uso dell'architettura esagonale nell'implementazione di un modello di dominio utilizzando una funzione Lambda.

Schema di pubblicazione-sottoscrizione

Intento

Il modello di pubblicazione-sottoscrizione, noto anche come modello pub-sub, è un modello di messaggistica che separa il mittente del messaggio (editore) dai destinatari interessati (sottoscrittori). Questo modello implementa comunicazioni asincrone pubblicando messaggi o eventi tramite un intermediario noto come broker di messaggi o router (infrastruttura di messaggi). Il modello di pubblicazione-sottoscrizione aumenta la scalabilità e la reattività dei mittenti, scaricando la responsabilità del recapito dei messaggi sull'infrastruttura dei messaggi in modo che il mittente possa concentrarsi sull'elaborazione principale dei messaggi.

Motivazione

Nelle architetture distribuite, i componenti del sistema spesso devono fornire informazioni ad altri componenti man mano che si verificano eventi all'interno del sistema. Lo schema di pubblicazione-sottoscrizione separa le preoccupazioni in modo che le applicazioni possano concentrarsi sulle proprie funzionalità principali mentre l'infrastruttura dei messaggi gestisce le responsabilità di comunicazione come il routing dei messaggi e la consegna affidabile. Il modello di pubblicazione-sottoscrizione consente la messaggistica asincrona per separare editore e sottoscrittori. Gli editori possono anche inviare messaggi all'insaputa dei sottoscrittori.

Applicabilità

Utilizza il modello di pubblicazione-sottoscrizione quando:

- È necessaria un'elaborazione parallela se un singolo messaggio ha flussi di lavoro diversi.
- Non sono richieste la trasmissione di messaggi a più sottoscrittori e le risposte in tempo reale da parte dei destinatari.
- Il sistema o l'applicazione possono tollerare l'eventuale coerenza dei dati o dello stato.
- L'applicazione o il componente deve comunicare con altre applicazioni o servizi che potrebbero utilizzare linguaggi, protocolli o piattaforme diversi.

Problemi e considerazioni

- Disponibilità dei sottoscrittori: l'editore non sa se i sottoscrittori stanno ascoltando e, in effetti, potrebbero non ascoltare. I messaggi pubblicati sono di natura temporanea e possono essere eliminati se i sottoscrittori non sono disponibili.
- Garanzia di consegna dei messaggi: in genere, il modello di pubblicazione-sottoscrizione non può garantire la consegna dei messaggi a tutti i tipi di sottoscrittori, sebbene alcuni servizi come Amazon Simple Notification Service (Amazon SNS) possano garantire la consegna una sola volta ad alcuni sottoinsiemi di sottoscrittori.
- Time to live (TTL): i messaggi hanno una durata e scadono se non vengono elaborati entro il periodo di tempo. Valuta la possibilità di aggiungere i messaggi pubblicati a una coda in modo che possano persistere e garantisci l'elaborazione oltre il periodo TTL.
- Pertinenza del messaggio: i producer possono impostare un intervallo di tempo per la pertinenza come parte dei dati del messaggio e il messaggio può essere eliminato dopo tale data. Valuta la possibilità di invitare i consumer a esaminare queste informazioni prima di decidere come elaborare il messaggio.
- Coerenza finale: c'è un ritardo tra il momento in cui il messaggio viene pubblicato e il momento in cui viene utilizzato dall'abbonato. Ciò potrebbe far sì che i datastore dei sottoscrittori diventino alla fine coerenti quando è richiesta una forte coerenza. L'eventuale coerenza potrebbe essere un problema anche quando producer e consumer richiedono un'interazione quasi in tempo reale.
- Comunicazione unidirezionale: il modello di pubblicazione-sottoscrizione è considerato unidirezionale. Le applicazioni che richiedono la messaggistica bidirezionale con un canale di sottoscrizione di ritorno dovrebbero prendere in considerazione l'utilizzo di uno schema di richiesta-risposta se è richiesta una risposta sincrona.
- Ordine dei messaggi: l'ordine dei messaggi non è garantito. Se i consumatori richiedono messaggi ordinati, ti consigliamo di utilizzare gli argomenti FIFO di Amazon SNS per garantire l'ordine.
- Duplicazione dei messaggi: in base all'infrastruttura di messaggistica, è possibile recapitare messaggi duplicati ai consumer. I consumer devono essere progettati per essere idempotenti nel gestire l'elaborazione di messaggi duplicati. In alternativa, utilizza gli argomenti FIFO di Amazon SNS per garantire la consegna esattamente una volta.
- Filtro dei messaggi: i consumer sono spesso interessati solo a un sottoinsieme dei messaggi pubblicati da un producer. Fornisci meccanismi per consentire agli abbonati di filtrare o limitare i messaggi che ricevono fornendo argomenti o filtri di contenuto.

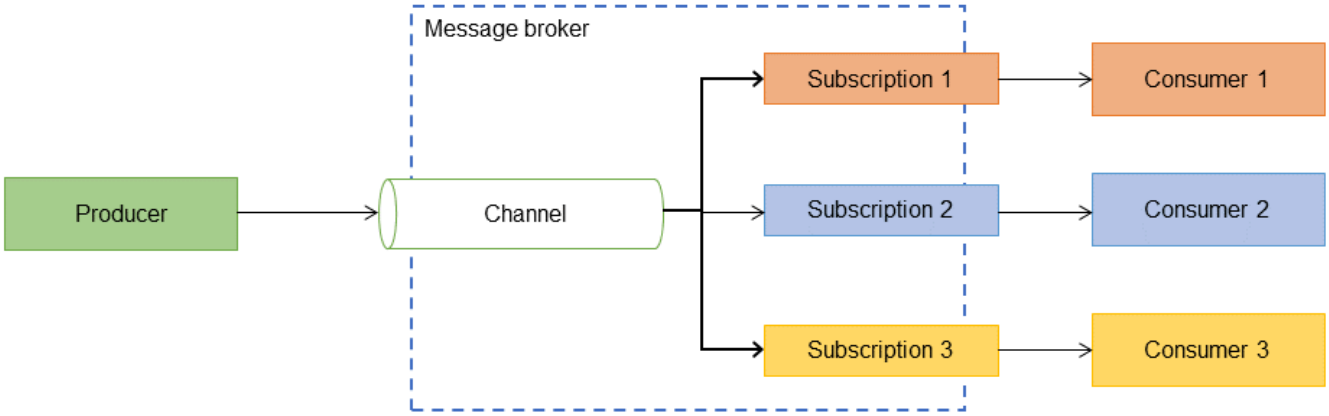
- Riproduzione dei messaggi: le funzionalità di riproduzione dei messaggi potrebbero dipendere dall'infrastruttura di messaggistica. È inoltre possibile fornire implementazioni personalizzate a seconda del caso d'uso.
- Code DLQ: in un sistema postale, un ufficio DL è una struttura per l'elaborazione della posta non recapitabile. Nella messaggistica pub/sub, una coda DLQ è una coda per i messaggi che non possono essere recapitati a un endpoint sottoscritto.

Implementazione

Architettura di alto livello

In un modello di pubblicazione/sottoscrizione, il sottosistema di messaggistica asincrono noto come broker di messaggi o router tiene traccia delle sottoscrizioni. Quando un producer pubblica un evento, l'infrastruttura di messaggistica invia un messaggio a ciascun consumer. Dopo che un messaggio è stato inviato agli abbonati, viene rimosso dall'infrastruttura dei messaggi in modo che non possa essere riprodotto e i nuovi abbonati non possano visualizzare l'evento. I broker di messaggi o i router separano il produttore di eventi dai consumer di messaggi in base a:

- Fornire al producer un canale di input per pubblicare eventi raggruppati in messaggi, utilizzando un formato di messaggio definito.
- Creazione di un singolo canale di output per sottoscrizione. Una sottoscrizione è la connessione del consumer, tramite la quale ascolta i messaggi relativi agli eventi associati a un canale di input specifico.
- Copia dei messaggi dal canale di input al canale di output per tutti i consumer quando l'evento viene pubblicato.



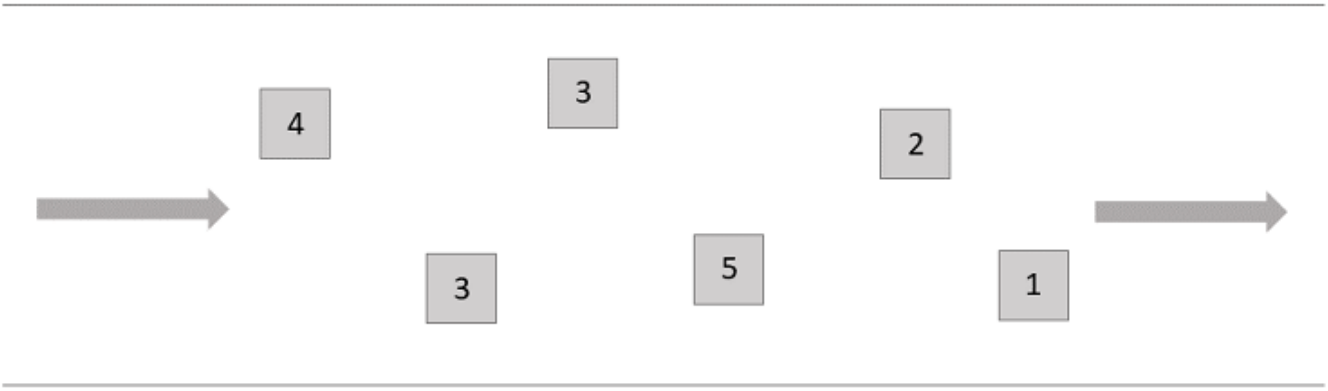
Implementazione tramite servizi AWS

Amazon SNS

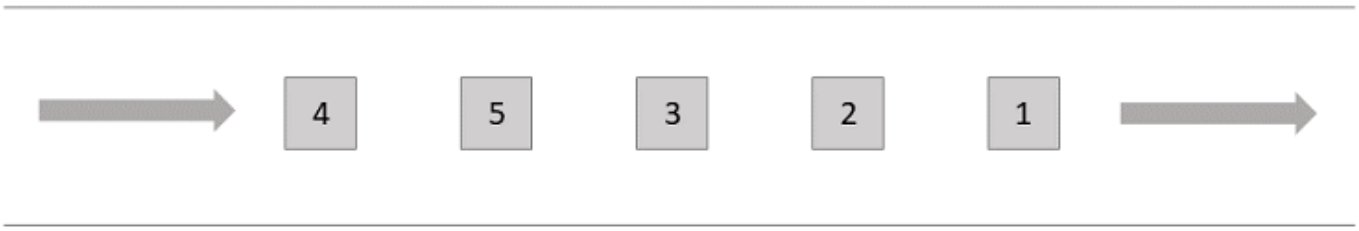
Amazon SNS è un servizio editore-abbonato completamente gestito che fornisce messaggistica da applicazione a applicazione (A2A) per disaccoppiare le applicazioni distribuite. Fornisce inoltre messaggistica da applicazione a persona (A2P) per l'invio di SMS, e-mail e altre notifiche push.

Amazon SNS offre due tipi di argomenti: standard e FIFO (first in, first out).

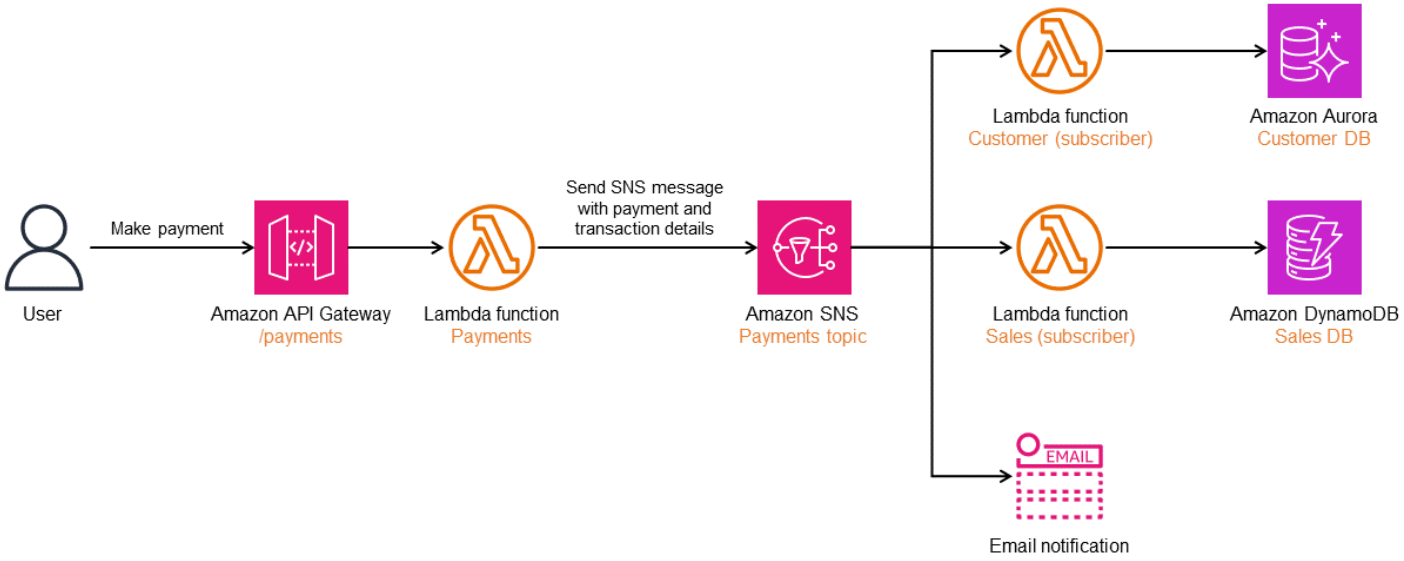
- Gli argomenti standard supportano un numero illimitato di messaggi al secondo e forniscono il massimo livello di ordinamento e deduplicazione.



- Gli argomenti FIFO forniscono un ordinamento e una deduplicazione rigorosi e supportano fino a 300 messaggi al secondo o 10 MB al secondo per argomento FIFO (a seconda di quale evento si verifica per primo).



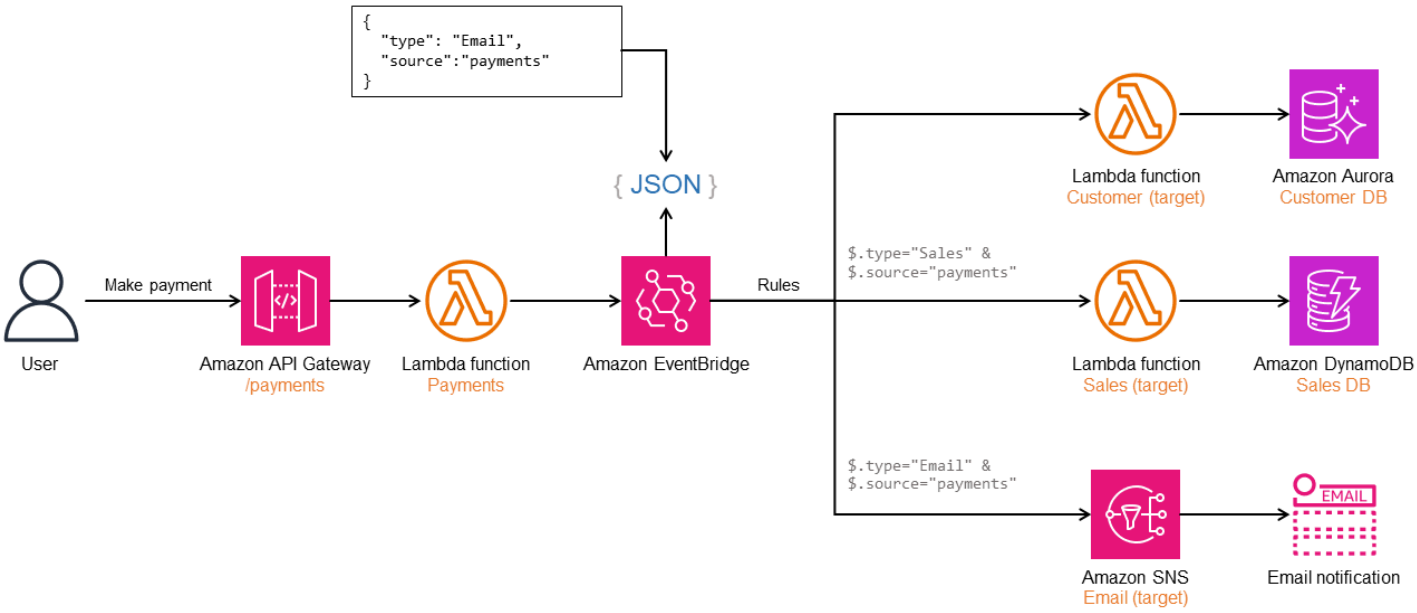
L'illustrazione seguente mostra come utilizzare Amazon SNS per implementare il modello di pubblicazione-sottoscrizione. Dopo che un utente ha effettuato un pagamento, la funzione Lambda `Payments` invia un messaggio SNS all'argomento SNS `Payments`. Questo argomento SNS ha tre sottoscrittori. Ogni sottoscrittore riceve una copia del messaggio e la elabora.



Amazon EventBridge

Puoi usare Amazon EventBridge quando hai bisogno di un routing più complesso dei messaggi da più producer attraverso protocolli diversi ai consumer abbonati o sottoscrizioni dirette e fan-out. EventBridge supporta anche il routing, il filtraggio, il sequenziamento e la suddivisione o l'aggregazione basati sui contenuti. Nella figura seguente, EventBridge viene utilizzato per creare una versione del modello pubblicazione-sottoscrizione in cui i sottoscrittori vengono definiti utilizzando le regole degli eventi. Dopo che un utente ha effettuato un pagamento, la funzione Lambda `Payments` invia un messaggio a EventBridge utilizzando il router di eventi predefinito basato su uno schema

personalizzato con tre regole che puntano a destinazioni diverse. Ogni microservizio elabora i messaggi ed esegue le azioni richieste.



Workshop

- Creazione di architetture basate sugli eventi in AWS
- Invio di notifiche di eventi fanout utilizzando Amazon Simple Queue Service (Amazon SQS) e Amazon Simple Notification Service (Amazon SNS)

Riferimenti del blog

- Scelta tra servizi di messaggistica per applicazioni serverless
- Progettazione di applicazioni serverless durevoli con DLQ per Amazon SNS, Amazon SQS, AWS Lambda
- Semplificazione della messaggistica pub/sub con il filtraggio dei messaggi di Amazon SNS

Contenuti correlati

- Funzionalità della messaggistica pub/sub

Riprova con schema di backoff

Intento

Lo schema Riprova con backoff migliora la stabilità dell'applicazione riprovando in modo trasparente le operazioni che hanno esito negativo a causa di errori transitori.

Motivazione

Nelle architetture distribuite, gli errori transitori possono essere causati dalla limitazione del servizio, dalla perdita temporanea della connettività di rete o dalla temporanea indisponibilità del servizio. La ripetizione automatica delle operazioni che falliscono a causa di questi errori transitori migliora l'esperienza utente e la resilienza delle applicazioni. Tuttavia, i tentativi frequenti possono sovraccaricare la larghezza di banda della rete e causare contese. Il backoff esponenziale è una tecnica in cui le operazioni vengono ripetute aumentando i tempi di attesa per un numero specificato di tentativi.

Applicabilità

Usa lo schema Riprova con backoff quando:

- I tuoi servizi spesso limitano la richiesta per evitare il sovraccarico, con conseguente429 Troppe richiesteeccezione al processo di chiamata.
- La rete partecipa in modo invisibile alle architetture distribuite e problemi temporanei di rete provocano guasti.
- Il servizio richiamato è temporaneamente non disponibile, con conseguenti errori. I tentativi frequenti possono causare il degrado del servizio a meno che non si introduca un timeout di backoff utilizzando questo schema.

Problemi e considerazioni

- Idempotenza: se più chiamate al metodo hanno lo stesso effetto di una singola chiamata sullo stato del sistema, l'operazione è considerata idempotente. Le operazioni devono essere idempotenti

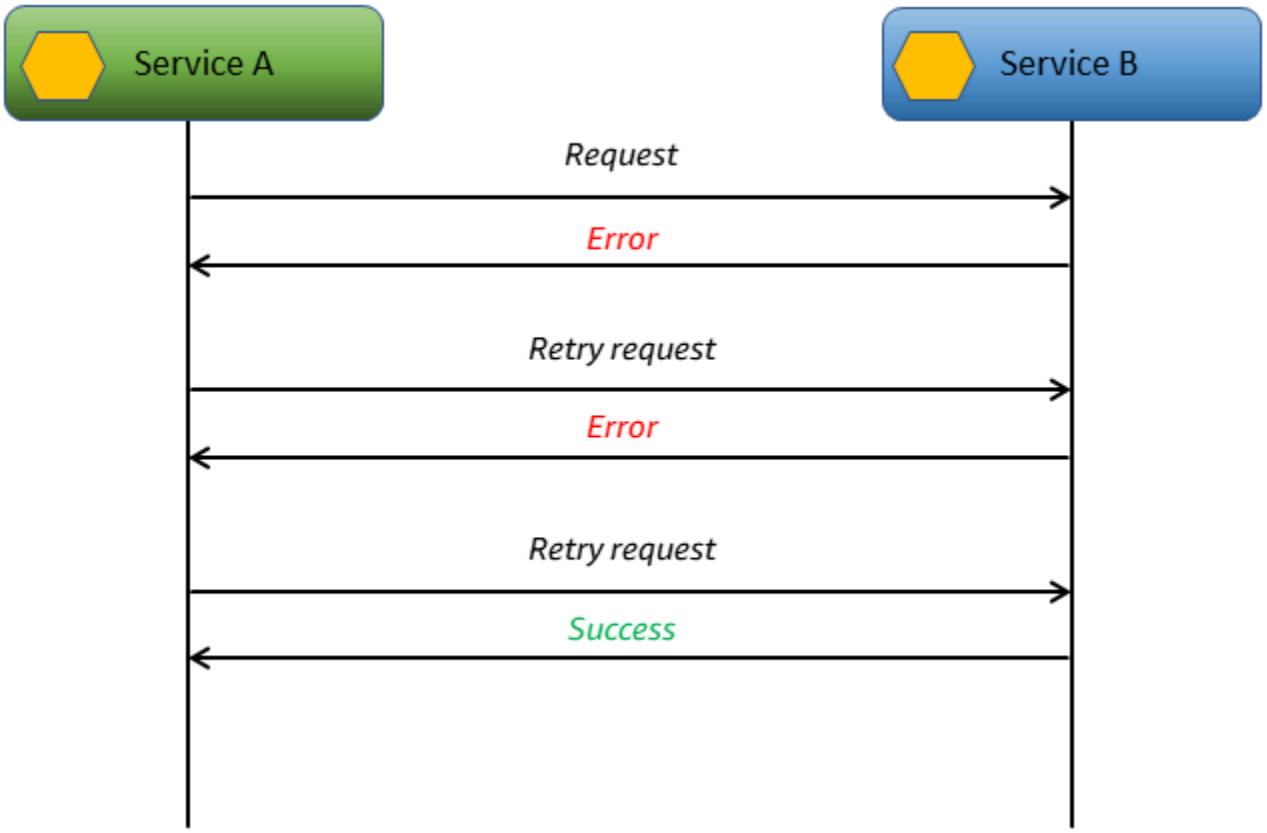
quando si utilizza lo schema Riprova con backoff. In caso contrario, gli aggiornamenti parziali potrebbero danneggiare lo stato del sistema.

- Larghezza di banda di rete: il degrado del servizio può verificarsi se troppi tentativi occupano la larghezza di banda della rete, con conseguenti tempi di risposta lenti.
- Scenari di fallimento rapido: Per gli errori non transitori, se è possibile determinare la causa del guasto, è più efficiente eseguire un guasto rapido utilizzando lo schema dell'interruttore automatico.
- Tasso di backoff: L'introduzione del backoff esponenziale può avere un impatto sul timeout del servizio, con conseguenti tempi di attesa più lunghi per l'utente finale.

Implementazione

Architettura di alto livello

Il diagramma seguente illustra come il servizio A può riprovare le chiamate al servizio B fino a quando non viene restituita una risposta corretta. Se il servizio B non restituisce una risposta corretta dopo alcuni tentativi, il servizio A può interrompere il tentativo e restituire un errore al chiamante.

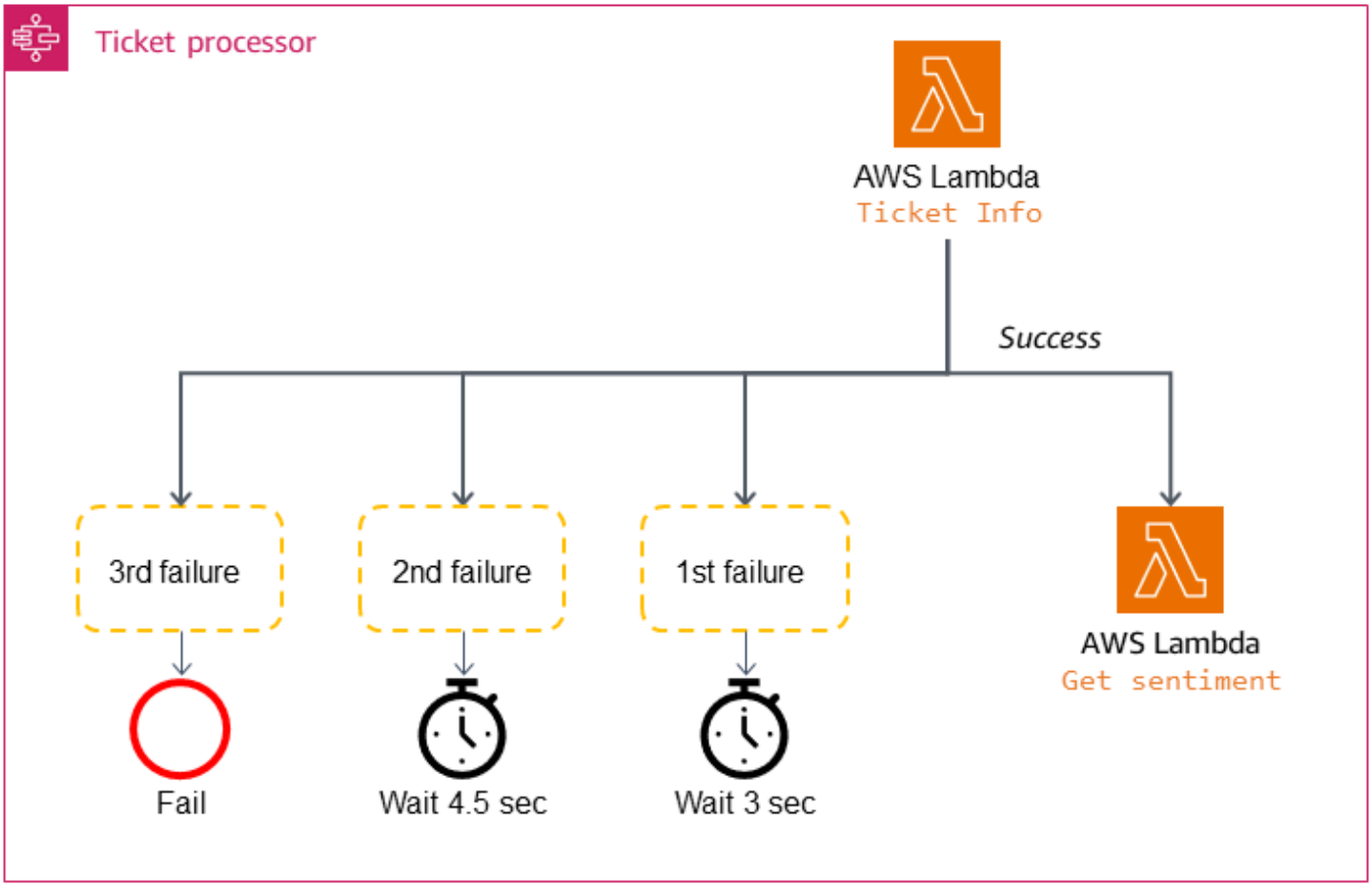


Implementazione utilizzandoAWSservizi

Il diagramma seguente mostra un flusso di lavoro di elaborazione dei ticket su una piattaforma di assistenza clienti. I ticket dei clienti insoddisfatti vengono accelerati aumentando automaticamente la priorità dei biglietti. La`Ticket info`La funzione Lambda estrae i dettagli del ticket e chiama il`Get sentiment`Funzione lambda. La`Get sentiment`La funzione Lambda verifica le opinioni dei clienti passando la descrizione aAmazon Comprehend(non mostrato).

Se la chiamata al`Get sentiment`La funzione Lambda fallisce, il flusso di lavoro riprova l'operazione tre volte.AWS Step Functionsconsente un backoff esponenziale consentendoti di configurare il valore di backoff.

In questo esempio, vengono configurati un massimo di tre tentativi con un moltiplicatore di incremento di 1,5 secondi. Se il primo tentativo avviene dopo 3 secondi, il secondo dopo 3 x 1,5 secondi = 4,5 secondi e il terzo dopo 4,5 x 1,5 secondi = 6,75 secondi. Se il terzo tentativo non va a buon fine, il flusso di lavoro ha esito negativo. La logica di backoff non richiede alcun codice personalizzato: viene fornita come configurazione daAWS Step Functions.



Codice di esempio

Il codice seguente mostra l'implementazione del pattern retry with 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);
  }
```

GitHubmagazzino

Per un'implementazione completa dell'architettura di esempio per questo modello, vedereGitHubrepository pressohttps://github.com/aws-samples/retry-with-backoff.

Contenuti correlati

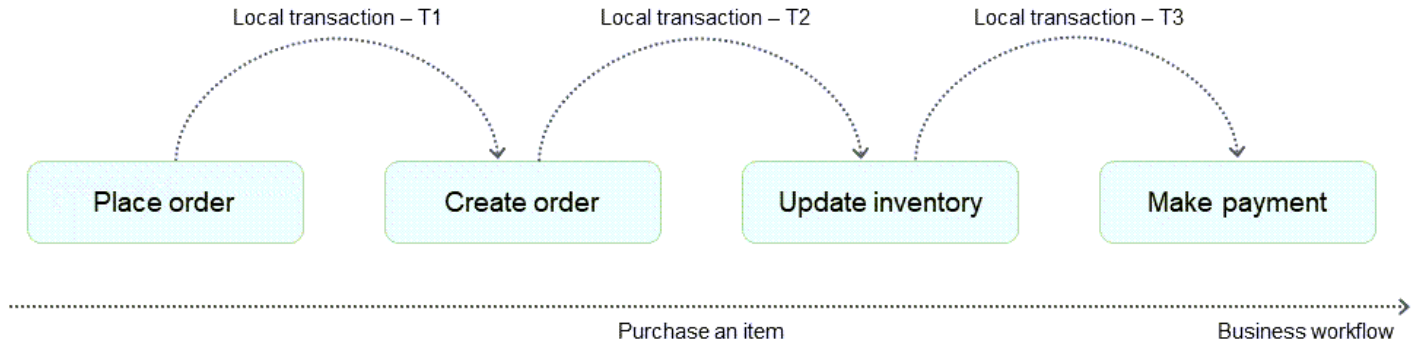
- Timeout, nuovi tentativi e retromarcia con jitter(Libreria di Amazon Builders)

Modelli saga

Una saga consiste in una sequenza di transazioni locali. Ogni transazione locale di una saga aggiorna il database e attiva la transazione locale successiva. Se una transazione non riesce, la saga esegue transazioni di compensazione per ripristinare le modifiche al database apportate dalle transazioni precedenti.

Questa sequenza di transazioni locali aiuta a realizzare un flusso di lavoro aziendale utilizzando i principi di continuazione e compensazione. Il principio di continuazione decide il ripristino successivo del flusso di lavoro, mentre il principio di compensazione decide il ripristino a ritroso. Se l'aggiornamento fallisce in qualsiasi fase della transazione, la saga pubblica un evento di continuazione (per riprovare la transazione) o di compensazione (per tornare allo stato precedente dei dati). Ciò garantisce il mantenimento dell'integrità dei dati e la coerenza in tutti i datastore.

Ad esempio, quando un utente acquista un libro da un rivenditore online, il processo consiste in una sequenza di transazioni, come la creazione dell'ordine, l'aggiornamento dell'inventario, il pagamento e la spedizione, che rappresenta un flusso di lavoro aziendale. Per completare questo flusso di lavoro, l'architettura distribuita emette una sequenza di transazioni locali per creare un ordine nel database degli ordini, aggiornare il database dell'inventario e aggiornare il database dei pagamenti. Quando il processo ha esito positivo, queste transazioni vengono richiamate in sequenza per completare il flusso di lavoro aziendale, come illustrato nel diagramma seguente. Tuttavia, se una di queste transazioni locali non riesce, il sistema deve essere in grado di decidere la fase successiva appropriata, ovvero un ripristino in avanti o un ripristino all'indietro.



I due scenari seguenti aiutano a determinare se il passaggio successivo è un ripristino in avanti o all'indietro:

- Errore a livello di piattaforma, in cui qualcosa va storto con l'infrastruttura sottostante e causa la non riuscita della transazione. In questo caso, il modello saga può eseguire un ripristino successivo ritentando la transazione locale e continuando il processo aziendale.
- Errore a livello di applicazione, in cui il servizio di pagamento non funziona a causa di un pagamento non valido. In questo caso, il modello saga può eseguire un ripristino a ritroso emettendo una transazione compensativa per aggiornare l'inventario e i database degli ordini e ripristinare lo stato precedente.

Il modello saga gestisce il flusso di lavoro aziendale e garantisce il raggiungimento dello stato finale desiderato attraverso il ripristino successivo. In caso di errori, ripristina le transazioni locali utilizzando il ripristino a ritroso per evitare problemi di coerenza dei dati.

Il modello saga ha due varianti: coreografia e orchestrazione.

Coreografia saga

Il modello coreografico della saga dipende dagli eventi pubblicati dai microservizi. I partecipanti alla saga (microservizi) sottoscrivono gli eventi e agiscono in base ai fattori scatenanti dell'evento. Ad esempio, il servizio ordini nel diagramma seguente emette un evento `OrderPlaced`. Il servizio di inventario sottoscrive quell'evento e aggiorna l'inventario quando viene emesso l'evento `OrderPlaced`. Allo stesso modo, i servizi per i partecipanti agiscono in base al contesto dell'evento emesso.

La coreografia è adatta quando ci sono solo pochi partecipanti alla saga ed è necessaria un'implementazione semplice senza alcun punto di errore. Quando vengono aggiunti più partecipanti, diventa più difficile tenere traccia delle dipendenze tra i vari partecipanti utilizzando questo schema.

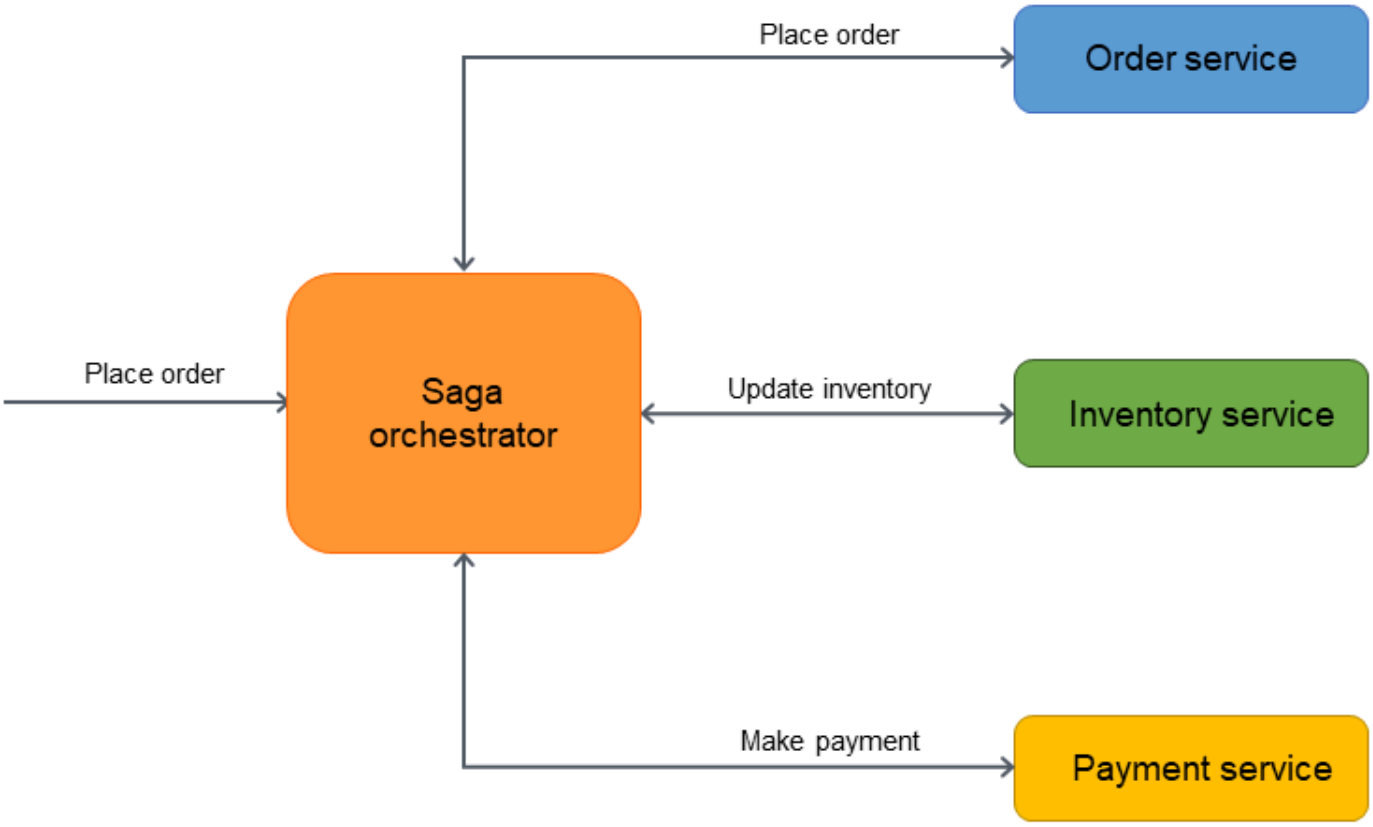
Per una revisione dettagliata, consulta la sezione Coreografia Saga in questa guida.

Orchestrazione saga

Il modello di orchestrazione saga ha un coordinatore centrale chiamato orchestratore. L'orchestratore saga gestisce e coordina l'intero ciclo di vita delle transazioni. È a conoscenza della serie di passaggi

da eseguire per completare la transazione. Per eseguire un passaggio, invia un messaggio al microservizio partecipante affinché esegua l'operazione. Il microservizio partecipante completa l'operazione e invia a sua volta un messaggio all'orchestratore. In base al messaggio ricevuto, l'orchestratore decide quale microservizio eseguire successivamente nella transazione.

Il modello di orchestrazione saga è adatto quando ci sono molti partecipanti ed è richiesto un accoppiamento libero tra i partecipanti alla saga. L'orchestratore incapsula la complessità della logica rendendo i partecipanti accoppiati in modo debole. Tuttavia, l'orchestratore può diventare un singolo punto di errore perché controlla l'intero flusso di lavoro.



Per una revisione dettagliata, consulta la sezione Orchestrazione Saga in questa guida.

Modello coreografico saga

Intento

Il modello coreografico saga aiuta a preservare l'integrità dei dati nelle transazioni distribuite che si estendono su più servizi utilizzando abbonamenti a eventi. In una transazione distribuita, è possibile

chiamare più servizi prima del completamento di una transazione. Quando i servizi archiviano i dati in datastore diversi, può essere difficile mantenere la coerenza dei dati tra questi.

Motivazione

Una transazione è una singola unità di lavoro che può comportare più passaggi, in cui tutti i passaggi vengono eseguiti completamente o non viene eseguito alcun passaggio, con il risultato di un datastore che mantiene lo stato coerente. I termini atomicità, consistenza, isolamento e durabilità (ACID) definiscono le proprietà di una transazione. I database relazionali forniscono transazioni ACID per mantenere la coerenza dei dati.

Per mantenere la coerenza in una transazione, i database relazionali utilizzano il metodo di commit a due fasi (2PC). Tale metodo consiste in una fase di preparazione e una fase di commit.

- Nella fase di preparazione, il processo di coordinamento richiede che i processi partecipanti alla transazione (partecipanti) si impegnino a confermare o annullare la transazione.
- Nella fase di commit, il processo di coordinamento richiede ai partecipanti di confermare la transazione. Se i partecipanti non riescono ad accettare di impegnarsi nella fase di preparazione, la transazione viene annullata.

Nei sistemi distribuiti che seguono uno schema di database-per-service progettazione, il commit in due fasi non è un'opzione. Questo perché ogni transazione è distribuita su diversi database e non esiste un unico controller in grado di coordinare un processo simile al commit in due fasi nei datastore relazionali. In questo caso, una soluzione consiste nell'utilizzare il modello coreografico saga.

Applicabilità

Usa il modello coreografico saga quando:

- Il sistema richiede l'integrità e la coerenza dei dati nelle transazioni distribuite che si estendono su più datastore.
- Il datastore (ad esempio, un database NoSQL) non fornisce 2PC per fornire transazioni ACID, è necessario aggiornare più tabelle all'interno di una singola transazione e implementare 2PC entro i confini dell'applicazione sarebbe un'attività complessa.
- Un processo di controllo centrale che gestisce le transazioni dei partecipanti potrebbe diventare un singolo punto di errore.
- I partecipanti alla saga sono servizi indipendenti e devono essere accoppiati in maniera debole.

- Esiste una comunicazione tra contesti limitati in un dominio aziendale.

Problemi e considerazioni

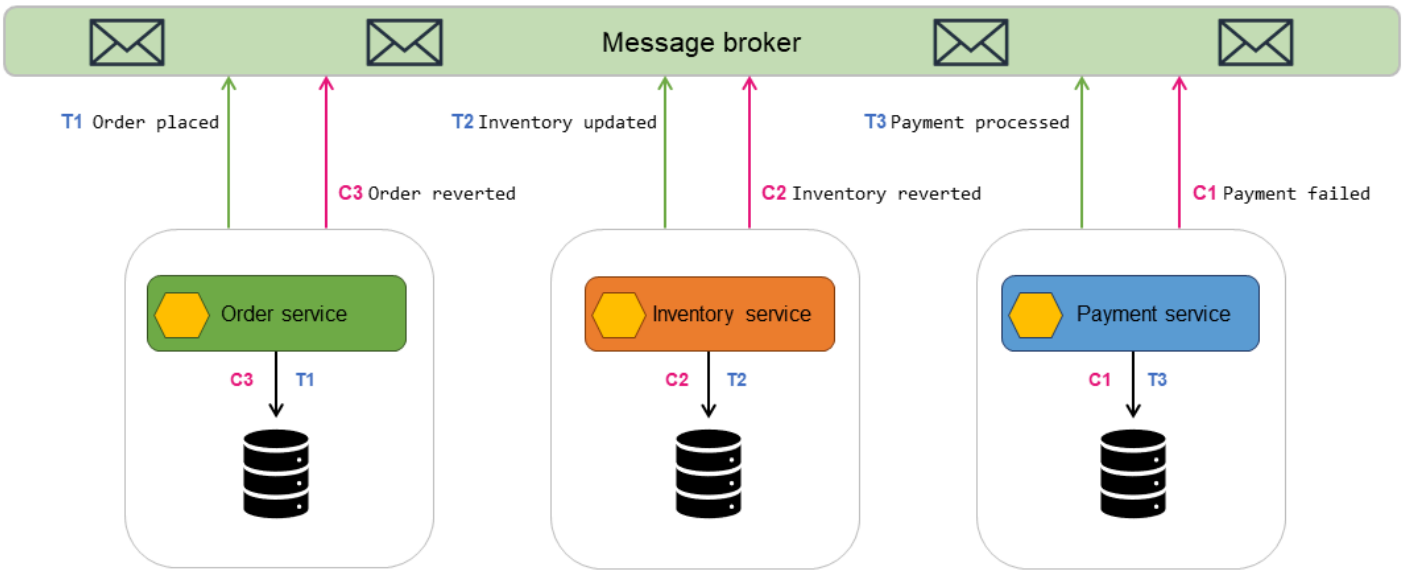
- Complessità: con l'aumentare del numero di microservizi, la coreografia saga può diventare difficile da gestire a causa del numero di interazioni tra i microservizi. Inoltre, le transazioni di compensazione e i nuovi tentativi aggiungono complessità al codice dell'applicazione, il che può aumentare la necessità di manutenzione. La coreografia è adatta quando ci sono solo pochi partecipanti alla saga ed è necessaria un'implementazione semplice senza alcun punto di errore. Quando vengono aggiunti più partecipanti, diventa più difficile tenere traccia delle dipendenze tra i vari partecipanti utilizzando questo schema.
- Implementazione resiliente: nella coreografia saga, è più difficile implementare timeout, nuovi tentativi e altri modelli di resilienza a livello globale, rispetto all'orchestrazione saga. La coreografia deve essere implementata su singoli componenti anziché a livello di orchestratore.
- Dipendenze cicliche: i partecipanti utilizzano messaggi pubblicati l'uno dall'altro. Ciò potrebbe comportare dipendenze cicliche, con conseguente complessità del codice e sovraccarichi di manutenzione e possibili situazioni di stallo.
- Problema di doppia scrittura: il microservizio deve aggiornare atomicamente il database e pubblicare un evento. L'errore di una delle due operazioni potrebbe portare a uno stato incoerente. Un modo per risolvere questo problema consiste nell'utilizzare il modello di posta in uscita transazionale.
- Conservazione degli eventi: i partecipanti alla saga agiscono in base agli eventi pubblicati. È importante salvare gli eventi nell'ordine in cui si verificano per scopi di controllo, debug e riproduzione. Per mantenere gli eventi in un datastore nel caso in cui sia necessaria una riproduzione dello stato del sistema per ripristinare la coerenza dei dati, è possibile utilizzare il modello di approvvigionamento degli eventi. Gli archivi eventi possono essere utilizzati anche per scopi di controllo e risoluzione dei problemi perché riflettono ogni modifica del sistema.
- Coerenza finale: l'elaborazione sequenziale delle transazioni locali si traduce in una coerenza finale, il che può rappresentare una sfida nei sistemi che richiedono una forte coerenza. È possibile risolvere questo problema impostando le aspettative dei team aziendali in merito al modello di consistenza o rivalutare il caso d'uso e passare a un datastore che garantisca una forte coerenza.
- Idempotenza: i partecipanti a saga devono essere idempotenti per consentire l'esecuzione ripetuta in caso di guasti transitori causati da arresti imprevisti e guasti dell'orchestratore.

- Isolamento delle transazioni: il modello saga non include l'isolamento delle transazioni, che è una delle quattro proprietà delle transazioni ACID. Il grado di isolamento di una transazione determina quanto o quanto poco altre transazioni simultanee possono influire sui dati su cui operano. L'orchestrazione simultanea delle transazioni può portare a dati obsoleti. Consigliamo di utilizzare il blocco semantico per gestire tali scenari.
- Osservabilità: l'osservabilità si riferisce alla registrazione e al tracciamento dettagliati per risolvere i problemi durante il processo di implementazione e orchestrazione. Ciò diventa importante quando il numero di partecipanti alla saga aumenta, con conseguenti complessità nel debug. nd-to-end Il monitoraggio e la reportistica elettronica sono più difficili da realizzare nella coreografia delle saga, rispetto all'orchestrazione delle saga.
- Problemi di latenza: le transazioni compensative possono aggiungere latenza al tempo di risposta complessivo quando la saga è composta da più passaggi. Se le transazioni effettuano chiamate sincrone, ciò può aumentare ulteriormente la latenza.

Implementazione

Architettura di alto livello

Nel seguente diagramma di architettura, l'orchestratore saga ha tre partecipanti: il servizio ordini, il servizio di inventario e il servizio di pagamento. Per completare la transazione sono necessari tre passaggi: T1, T2 e T3. Tre transazioni compensative consentono di ripristinare i dati allo stato iniziale: C1, C2 e C3.



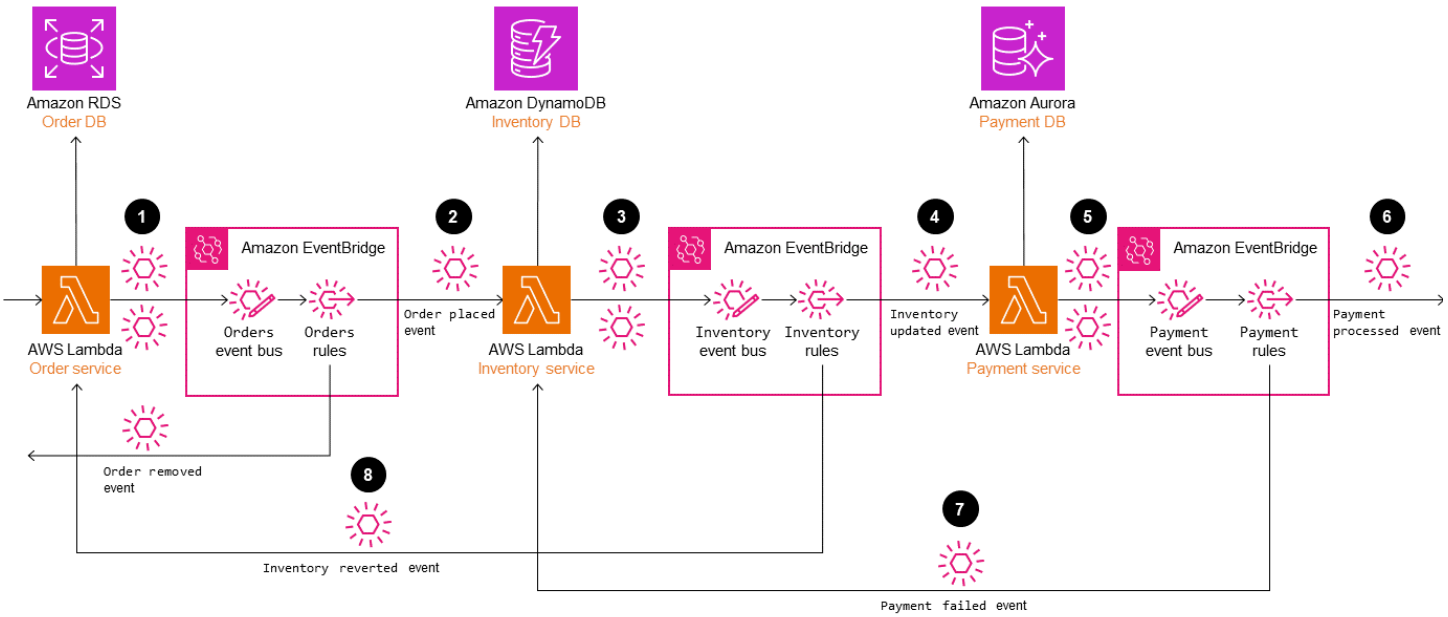
- Il servizio ordini esegue una transazione locale, T1, che aggiorna atomicamente il database e pubblica un messaggio `Order placed` sul broker di messaggi.
- Il servizio di inventario sottoscrive i messaggi del servizio ordini e riceve il messaggio che indica che è stato creato un ordine.
- Il servizio di inventario esegue una transazione locale, T2, che aggiorna atomicamente il database e pubblica un messaggio `Inventory updated` sul broker di messaggi.
- Il servizio di pagamento sottoscrive i messaggi del servizio di inventario e riceve il messaggio che l'inventario è stato aggiornato.
- Il servizio di pagamento esegue una transazione locale, T3, che aggiorna atomicamente il database con i dettagli del pagamento e pubblica un messaggio `Payment processed` sul broker di messaggi.
- Se il pagamento non riesce, il servizio di pagamento esegue una transazione compensativa, C1, che annulla atomicamente il pagamento nel database e pubblica un messaggio `Payment failed` sul broker di messaggi.
- Le transazioni compensative C2 e C3 vengono eseguite per ripristinare la coerenza dei dati.

Implementazione tramite servizi AWS

Puoi implementare il pattern coreografico della saga utilizzando Amazon. EventBridge EventBridgeutilizza gli eventi per connettere i componenti dell'applicazione. Elabora gli eventi tramite bus o pipe di eventi. Un router di eventi è un router che riceve eventi e li invia a nessuna o a più destinazioni o target. Le regole associate al router di eventi valutano gli eventi man mano che arrivano e li inviano alle destinazioni per l'elaborazione.

Nella seguente architettura:

- I microservizi (servizio ordini, servizio di inventario e servizio di pagamento) sono implementati come funzioni Lambda.
- Esistono tre EventBridge bus personalizzati: `Orders` event bus, `Inventory` event bus e `Payment` event bus.
- Le regole `Orders`, le regole `Inventory` e le regole `Payment` corrispondono agli eventi che vengono inviati al router di eventi corrispondente e richiamano le funzioni Lambda.



In uno scenario di successo, quando viene effettuato un ordine:

1. Il servizio ordini elabora la richiesta e invia l'evento al router di eventi `Orders`.
2. Le regole `Orders` corrispondono agli eventi e avviano il servizio di inventario.
3. Il servizio di inventario aggiorna l'inventario e invia l'evento al router di eventi `Inventory`.
4. Le regole `Inventory` corrispondono agli eventi e avviano il servizio di pagamento.
5. Il servizio di pagamento elabora il pagamento e invia l'evento al router di eventi `Payment`.
6. Le regole `Payment` corrispondono agli eventi e inviano la notifica dell'evento `Payment processed` all'ascoltatore.

   In alternativa, quando si verifica un problema nell'elaborazione degli ordini, le EventBridge regole avviano le transazioni compensative per ripristinare gli aggiornamenti dei dati per mantenere la coerenza e l'integrità dei dati.
7. Se il pagamento non riesce, le regole `Payment` elaborano l'evento e avviano il servizio di inventario.Il servizio di inventario esegue quindi le transazioni compensative per ripristinare l'inventario.
8. Una volta ripristinato l'inventario, il servizio di inventario invia l'evento `Inventory reverted` al router di eventi `Inventory`.Questo evento viene elaborato dalle regole `Inventory`.Avvia il servizio ordini, che esegue la transazione compensativa per rimuovere l'ordine.

Contenuti correlati

- Modello di orchestrazione saga
- Modello transazionale di posta in uscita
- Nuovo tentativo con schema di backoff

Modello di orchestrazione saga

Intento

Il modello di orchestrazione saga utilizza un coordinatore centrale (orchestratore) per aiutare a preservare l'integrità dei dati nelle transazioni distribuite che si estendono su più servizi. In una transazione distribuita, è possibile chiamare più servizi prima del completamento di una transazione. Quando i servizi archiviano i dati in datastore diversi, può essere difficile mantenere la coerenza dei dati tra questi.

Motivazione

Una transazione è una singola unità di lavoro che può comportare più passaggi, in cui tutti i passaggi vengono eseguiti completamente o non viene eseguito alcun passaggio, con il risultato di un datastore che mantiene lo stato coerente. I termini atomicità, consistenza, isolamento e durabilità (ACID) definiscono le proprietà di una transazione. I database relazionali forniscono transazioni ACID per mantenere la coerenza dei dati.

Per mantenere la coerenza in una transazione, i database relazionali utilizzano il metodo di commit a due fasi (2PC). Tale metodo consiste in una fase di preparazione e una fase di commit.

- Nella fase di preparazione, il processo di coordinamento richiede che i processi partecipanti alla transazione (partecipanti) si impegnino a confermare o annullare la transazione.
- Nella fase di commit, il processo di coordinamento richiede ai partecipanti di confermare la transazione. Se i partecipanti non riescono ad accettare di impegnarsi nella fase di preparazione, la transazione viene annullata.

Nei sistemi distribuiti che seguono uno schema di database-per-service progettazione, il commit in due fasi non è un'opzione. Questo perché ogni transazione è distribuita su diversi database e non

esiste un unico controller in grado di coordinare un processo simile al commit in due fasi nei datastore relazionali. In questo caso, una soluzione consiste nell'utilizzare il modello di orchestrazione saga.

Applicabilità

Usa il modello di orchestrazione saga quando:

- Il sistema richiede l'integrità e la coerenza dei dati nelle transazioni distribuite che si estendono su più datastore.
- Il datastore non fornisce 2PC per fornire transazioni ACID e implementare 2PC entro i confini dell'applicazione è un'attività complessa.
- Disponi di database NoSQL, che non forniscono transazioni ACID, e devi aggiornare più tabelle all'interno di una singola transazione.

Problemi e considerazioni

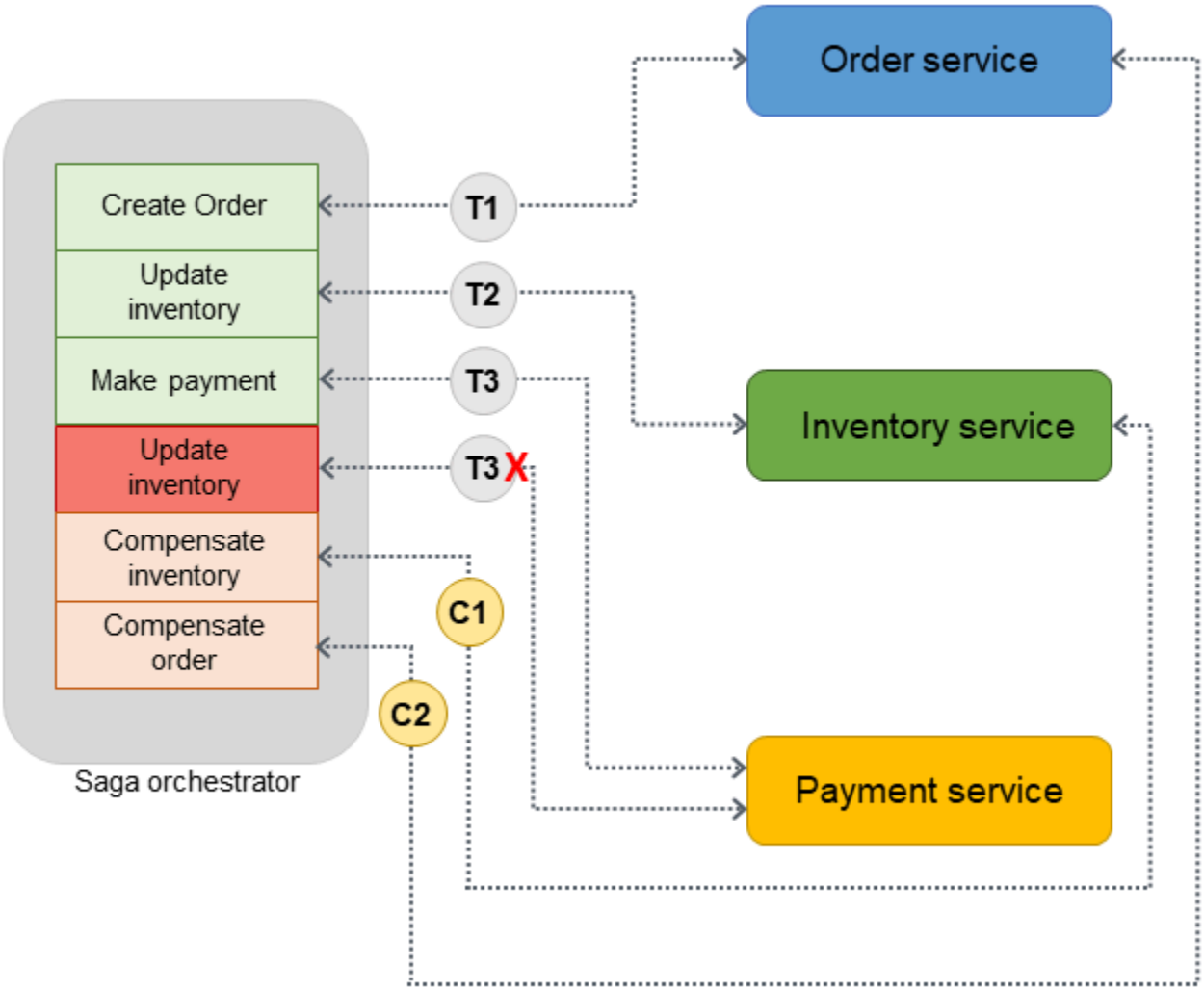
- Complessità: le transazioni compensative e i nuovi tentativi aggiungono complessità al codice dell'applicazione, il che può comportare costi di manutenzione.
- Coerenza finale: l'elaborazione sequenziale delle transazioni locali si traduce in una coerenza finale, il che può rappresentare una sfida nei sistemi che richiedono una forte coerenza. È possibile risolvere questo problema impostando le aspettative dei team aziendali in merito al modello di consistenza o passando a un datastore che garantisca una forte coerenza.
- Idempotenza: i partecipanti a saga devono essere idempotenti per consentire l'esecuzione ripetuta in caso di guasti transitori causati da arresti imprevisti e guasti dell'orchestratore.
- Isolamento delle transazioni: la saga non dispone dell'isolamento delle transazioni. L'orchestrazione simultanea delle transazioni può portare a dati obsoleti. Consigliamo di utilizzare il blocco semantico per gestire tali scenari.
- Osservabilità: l'osservabilità si riferisce alla registrazione e al tracciamento dettagliati per risolvere i problemi durante il processo di esecuzione e orchestrazione. Ciò diventa importante quando il numero di partecipanti alla saga aumenta, con conseguenti complessità nel debug.
- Problemi di latenza: le transazioni compensative possono aggiungere latenza al tempo di risposta complessivo quando la saga è composta da più passaggi. In questi casi, evita le chiamate sincrone.

- Singolo punto di errore: l'orchestratore può diventare un singolo punto di errore perché coordina l'intera transazione. In alcuni casi, a causa di questo problema, si preferisce il modello coreografico della saga.

Implementazione

Architettura di alto livello

Nel seguente diagramma di architettura, l'orchestratore saga ha tre partecipanti: il servizio ordini, il servizio di inventario e il servizio di pagamento. Per completare la transazione sono necessari tre passaggi: T1, T2 e T3. L'orchestratore saga conosce i passaggi e li esegue nell'ordine richiesto. Quando la fase T3 non riesce (errore di pagamento), l'orchestratore esegue le transazioni compensative C1 e C2 per ripristinare i dati allo stato iniziale.



Puoi usare AWS Step Functions per implementare l'orchestrazione saga quando la transazione viene distribuita su più database.

Implementazione tramite servizi AWS

La soluzione di esempio utilizza il flusso di lavoro standard di Step Functions per implementare il modello di orchestrazione saga.

Quando un cliente chiama l'API, viene richiamata la funzione Lambda, dove avviene la fase di pre-elaborazione. La funzione avvia il flusso di lavoro Step Functions per avviare l'elaborazione della transazione distribuita. Se la pre-elaborazione non è richiesta, puoi avviare il flusso di lavoro Step Functions direttamente da Gateway API senza utilizzare la funzione Lambda.

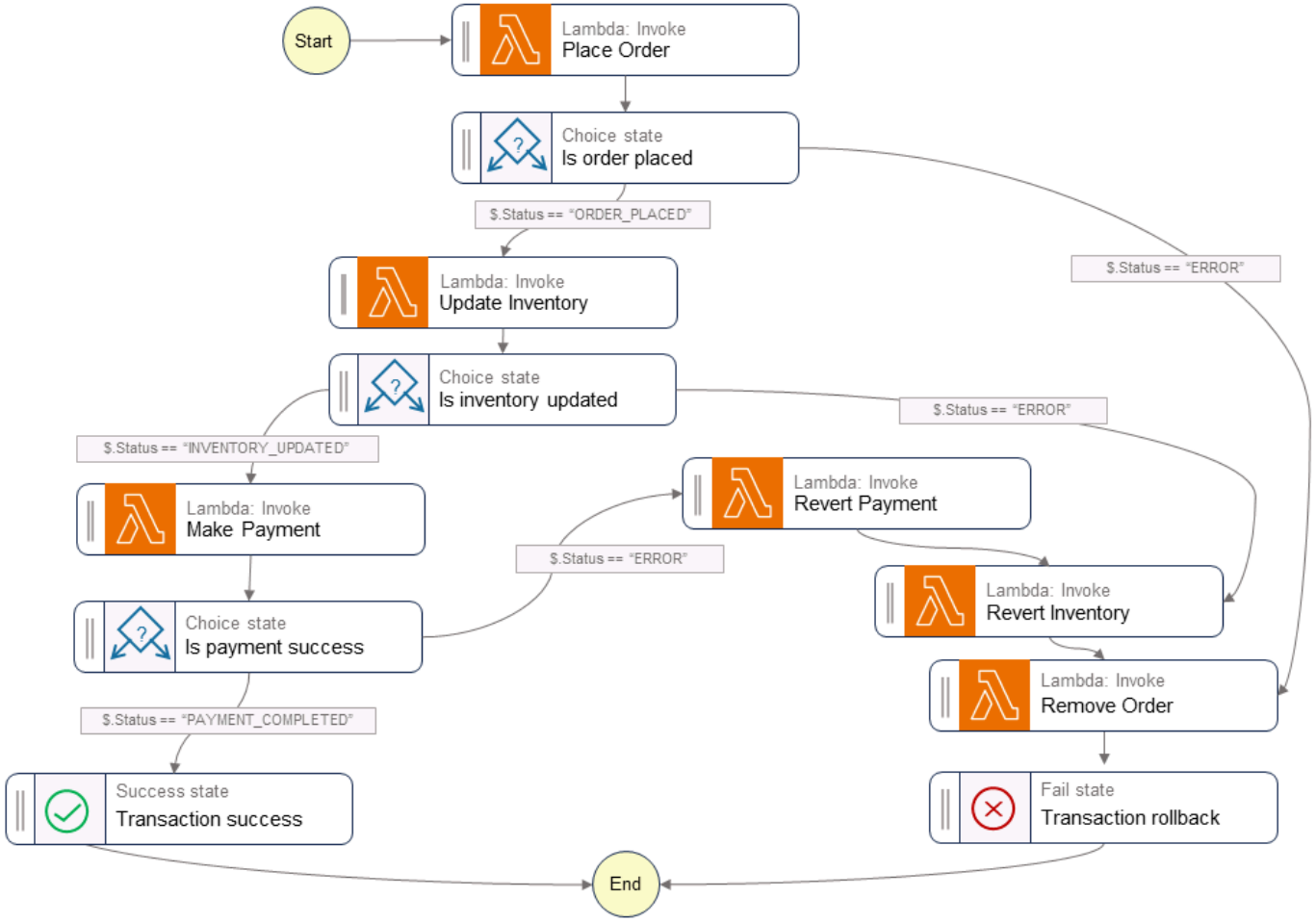
L'uso di Step Functions mitiga il problema del singolo punto di errore, relativo all'implementazione del modello di orchestrazione saga. Step Functions ha una tolleranza agli errori integrata e mantiene la capacità di servizio in più zone di disponibilità in ogni Regione AWS per proteggere le applicazioni dai guasti di singoli computer o data center. Ciò contribuisce a garantire un'elevata disponibilità sia per il servizio stesso che per il flusso di lavoro dell'applicazione su cui opera.

Il flusso di lavoro Step Functions

La macchina a stati Step Functions consente di configurare i requisiti del flusso di controllo basato sulle decisioni per l'implementazione del modello. Il flusso di lavoro Step Functions richiama i singoli servizi per l'inserimento degli ordini, l'aggiornamento dell'inventario e l'elaborazione dei pagamenti per completare la transazione e invia una notifica dell'evento per un'ulteriore elaborazione. Il flusso di lavoro Step Functions funge da orchestratore per coordinare le transazioni. Se il flusso di lavoro contiene errori, l'orchestratore esegue le transazioni compensative per garantire che l'integrità dei dati sia mantenuta tra i servizi.

Il seguente diagramma mostra i passaggi eseguiti all'interno del flusso di lavoro Step Functions. I passaggi `Place Order`, `Update Inventory` e `Make Payment` indicano il percorso riuscito. L'ordine viene effettuato, l'inventario viene aggiornato e il pagamento viene elaborato prima che al chiamante venga restituito lo stato `Success`.

Le funzioni Lambda `Revert Payment`, `Revert Inventory` e `Remove Order` indicano le transazioni compensative che l'orchestratore esegue quando una qualsiasi fase del flusso di lavoro non riesce. Se il flusso di lavoro fallisce durante la fase `Update Inventory`, l'orchestratore chiama le fasi `Revert Inventory` e `Remove Order` prima di restituire uno stato `Fail` al chiamante. Queste transazioni compensative garantiscono il mantenimento dell'integrità dei dati. L'inventario torna al livello originale e l'ordine viene ripristinato.



Codice di esempio

Il seguente codice di esempio mostra come è possibile creare un orchestratore saga utilizzando Step Functions. Per visualizzare il codice completo, consultate il GitHubrepository di questo esempio.

Definizioni di processi

```
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);
```

Step Functions e definizioni macchine a stati

```
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 repository

Per un'implementazione completa dell'architettura di esempio per questo pattern, consultate il GitHub repository all'indirizzo https://github.com/aws-samples/. saga-orchestration-netcore-blog

Riferimenti del blog

- Creazione di un'applicazione distribuita serverless utilizzando il modello di orchestrazione saga

Contenuti correlati

- Modello coreografico saga
- Modello transazionale di posta in uscita

Video

Il video seguente illustra come implementare il pattern di orchestrazione della saga utilizzando. AWS Step Functions

Modello Scatter-gather

Intento

Il pattern scatter-collect è uno schema di routing dei messaggi che prevede la trasmissione di richieste simili o correlate a più destinatari e l'aggregazione delle relative risposte in un unico messaggio utilizzando un componente chiamato aggregatore. Questo modello consente di ottenere la parallelizzazione, riduce la latenza di elaborazione e gestisce la comunicazione asincrona. È semplice implementare il pattern scatter-collect utilizzando un approccio sincrono, ma un approccio più potente prevede l'implementazione come routing dei messaggi nelle comunicazioni asincrone, con o senza un servizio di messaggistica.

Motivazione

Nell'elaborazione delle applicazioni, una richiesta che potrebbe richiedere molto tempo per essere elaborata in sequenza può essere suddivisa in più richieste elaborate in parallelo. È inoltre possibile inviare richieste a più sistemi esterni tramite chiamate API per ottenere una risposta. Il pattern scatter-gather è utile quando è necessario un input da più fonti. Scatter-gather aggrega i risultati per aiutarvi a prendere una decisione informata o a selezionare la risposta migliore per la richiesta.

Il pattern scatter-gather si compone di due fasi, come suggerisce il nome:

- La fase di dispersione elabora il messaggio di richiesta e lo invia a più destinatari in parallelo. Durante questa fase, l'applicazione distribuisce le richieste in tutta la rete e continua a funzionare senza attendere risposte immediate.
- Durante la fase di raccolta, l'applicazione raccoglie le risposte dai destinatari e le filtra o le combina in una risposta unificata. Una volta raccolte tutte le risposte, è possibile aggregarle in un'unica risposta oppure scegliere la migliore per un'ulteriore elaborazione.

Applicabilità

Usa lo schema scatter-gather quando:

- Hai intenzione di aggregare e consolidare i dati provenienti da varie API per creare una risposta accurata. Il modello consolida le informazioni provenienti da fonti diverse in un insieme coeso. Ad

esempio, un sistema di prenotazione può richiedere a più destinatari di ricevere preventivi da più partner esterni.

- La stessa richiesta deve essere inviata a più destinatari contemporaneamente per completare una transazione. Ad esempio, puoi utilizzare questo modello per interrogare i dati di inventario in parallelo per verificare la disponibilità di un prodotto.
- Desiderate implementare un sistema affidabile e scalabile in cui sia possibile ottenere il bilanciamento del carico distribuendo le richieste tra più destinatari. Se un destinatario fallisce o subisce un carico di lavoro elevato, gli altri destinatari possono comunque elaborare le richieste.
- Desideri ottimizzare le prestazioni durante l'implementazione di query complesse che coinvolgono più fonti di dati. È possibile distribuire la query nei database pertinenti, raccogliere i risultati parziali e combinarli in una risposta completa.
- Stai implementando un tipo di elaborazione map-reduce in cui la richiesta di dati viene indirizzata a più endpoint di elaborazione dei dati per lo sharding e la replica. I risultati parziali vengono filtrati e combinati per comporre la risposta corretta.
- Desideri distribuire le operazioni di scrittura su uno spazio di chiavi di partizione in carichi di lavoro che richiedono molta scrittura nei database chiave-valore. L'aggregatore legge i risultati interrogando i dati in ogni shard e quindi li consolida in un'unica risposta.

Problemi e considerazioni

- Tolleranza agli errori: questo modello si basa su più destinatari che lavorano in parallelo, quindi è essenziale gestire gli errori con garbo. Per mitigare l'impatto degli errori dei destinatari sull'intero sistema, è possibile implementare strategie come ridondanza, replica e rilevamento degli errori.
- Limiti di scalabilità orizzontale: all'aumentare del numero totale di nodi di elaborazione, aumenta anche il sovraccarico di rete associato. Ogni richiesta che implica la comunicazione sulla rete può aumentare la latenza e influire negativamente sui vantaggi della parallelizzazione.
- Ostacoli nei tempi di risposta: per le operazioni che richiedono l'elaborazione di tutti i destinatari prima del completamento dell'elaborazione finale, le prestazioni dell'intero sistema sono limitate dal tempo di risposta del destinatario più lento.
- Risposte parziali: quando le richieste vengono distribuite a più destinatari, alcuni destinatari possono scadere. In questi casi, l'implementazione dovrebbe comunicare al client che la risposta è incompleta. È inoltre possibile visualizzare i dettagli di aggregazione della risposta utilizzando un frontend dell'interfaccia utente.

- Coerenza dei dati: quando si elaborano dati tra più destinatari, è necessario considerare attentamente le tecniche di sincronizzazione dei dati e di risoluzione dei conflitti, per garantire che i risultati aggregati finali siano accurati e coerenti.

Implementazione

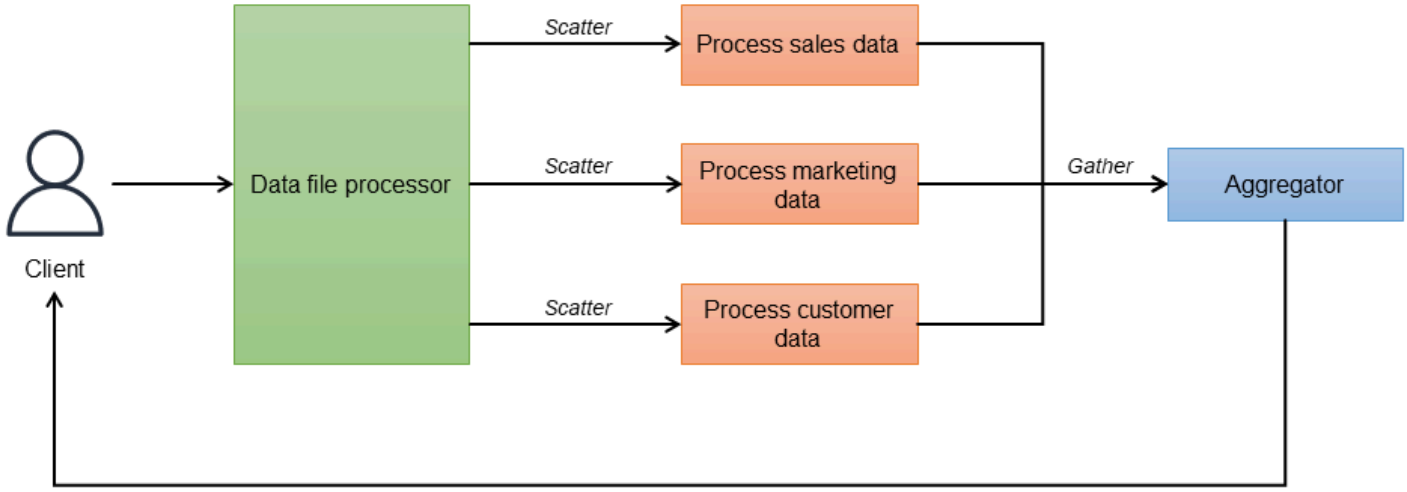
Architettura di alto livello

Il pattern scatter-collect utilizza un controller root per distribuire le richieste ai destinatari che le elaboreranno. Durante la fase di dispersione, questo pattern può utilizzare due meccanismi per inviare messaggi ai destinatari:

- Scatter by distribution: l'applicazione dispone di un elenco noto di destinatari che devono essere chiamati per ottenere i risultati. I destinatari possono essere processi diversi con funzioni uniche o un singolo processo che è stato ridimensionato per distribuire il carico di elaborazione. Se uno dei nodi di elaborazione scade o mostra ritardi nella risposta, il controller può ridistribuire l'elaborazione su un altro nodo.
- Scatter by auction: l'applicazione trasmette il messaggio ai destinatari interessati utilizzando uno schema di pubblicazione e sottoscrizione. In questo caso, i destinatari possono iscriversi al messaggio o recedere dall'abbonamento in qualsiasi momento.

Spargi per distribuzione

Nel metodo di distribuzione scatter by, il root controller divide la richiesta in entrata in attività indipendenti e le assegna ai destinatari disponibili (la fase di dispersione). Ogni destinatario (processo, contenitore o funzione Lambda) esegue il calcolo in modo indipendente e parallelo e produce una parte della risposta. Quando i destinatari completano le proprie attività, inviano le risposte a un aggregatore (la fase di raccolta). L'aggregatore combina le risposte parziali e restituisce il risultato finale al cliente. Il diagramma seguente illustra questo flusso di lavoro.

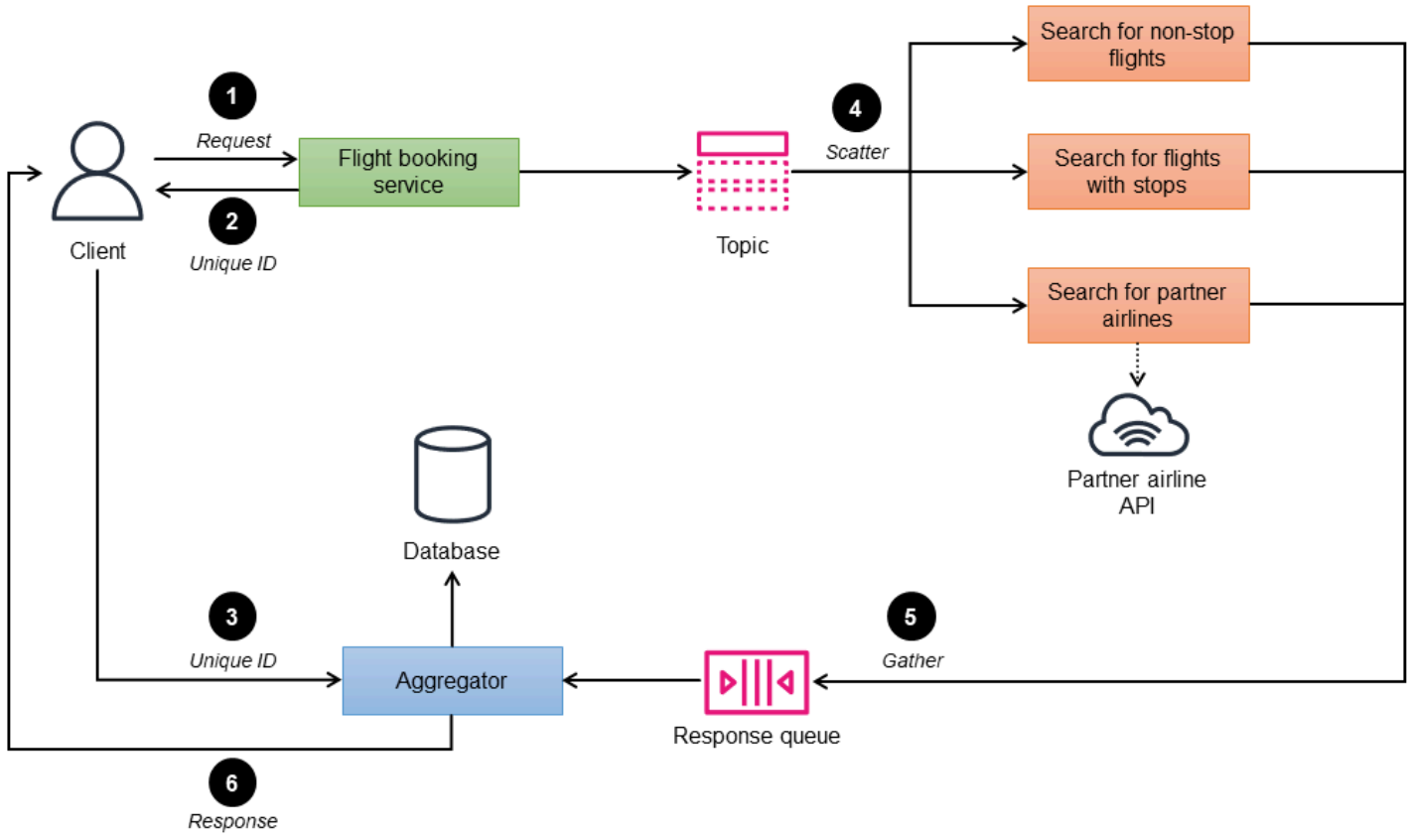


Il controller (elaboratore di file di dati) orchestra l'intero set di chiamate ed è a conoscenza di tutti gli endpoint di prenotazione da chiamare. Può configurare un parametro di timeout per ignorare le risposte che richiedono troppo tempo. Una volta inviate le richieste, l'aggregatore attende le risposte da ciascun endpoint. Per implementare la resilienza, ogni microservizio può essere distribuito con più istanze per il bilanciamento del carico. L'aggregatore ottiene i risultati, li combina in un unico messaggio di risposta e rimuove i dati duplicati prima dell'ulteriore elaborazione. Le risposte scadute vengono ignorate. Il controller può anche fungere da aggregatore anziché utilizzare un servizio di aggregazione separato.

Scatter per asta

Se il controllore non conosce i destinatari o se i destinatari sono associati in modo incerto, puoi utilizzare il metodo scatter by auction. In questo metodo, i destinatari sottoscrivono un argomento e il controller pubblica la richiesta sull'argomento. I destinatari pubblicano i risultati in una coda di risposta. Poiché il controller principale non conosce i destinatari, il processo di raccolta utilizza un aggregatore (un altro modello di messaggistica) per raccogliere le risposte e distillarle in un unico messaggio di risposta. L'aggregatore utilizza un ID univoco per identificare un gruppo di richieste.

Ad esempio, nel diagramma seguente, il metodo scatter by auction viene utilizzato per implementare un servizio di prenotazione di voli per il sito Web di una compagnia aerea. Il sito Web consente agli utenti di cercare e visualizzare voli del vettore della compagnia aerea e dei vettori dei suoi partner e deve visualizzare lo stato della ricerca in tempo reale. Il servizio di prenotazione voli è composto da tre microservizi di ricerca: voli diretti, voli con scali e compagnie aeree partner. La ricerca della compagnia aerea partner chiama gli endpoint API del partner per ottenere le risposte.

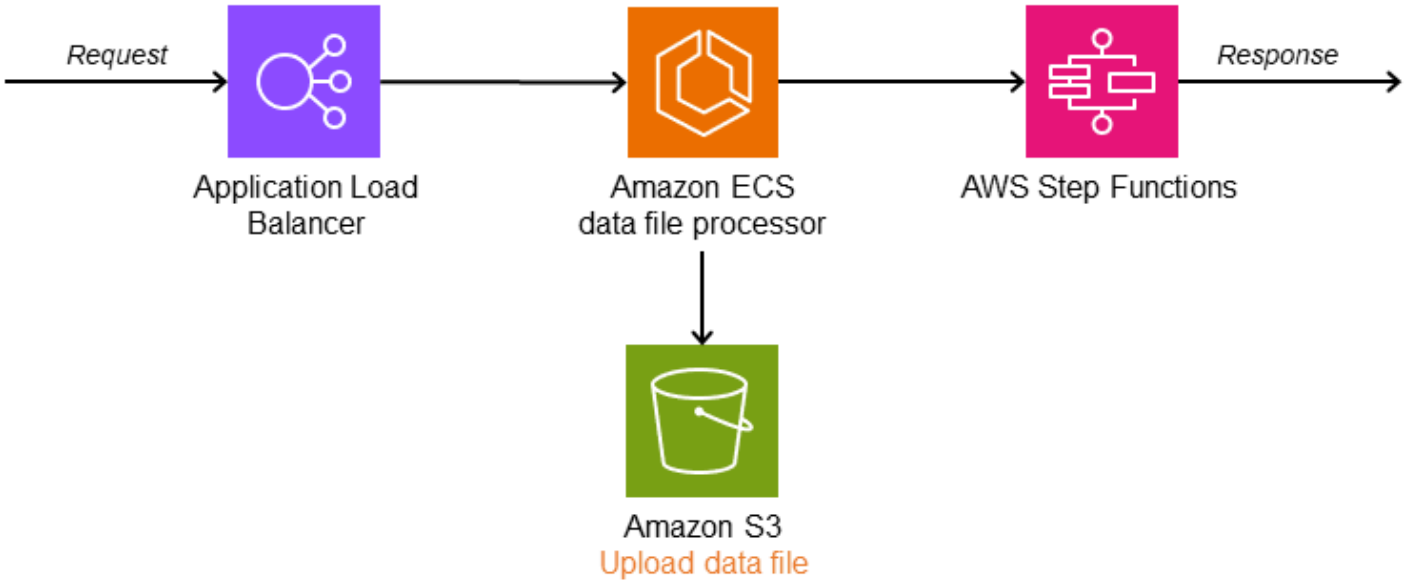


1. Il servizio di prenotazione dei voli (controller) accetta i criteri di ricerca inseriti dal cliente, elabora e pubblica la richiesta sull'argomento.
2. Il controller utilizza un ID univoco per identificare ogni gruppo di richieste.
3. Il client invia l'ID univoco all'aggregatore per la fase 6.
4. I microservizi di ricerca delle prenotazioni che hanno sottoscritto l'argomento di prenotazione ricevono la richiesta.
5. I microservizi elaborano la richiesta e restituiscono la disponibilità dei posti per i criteri di ricerca specificati in una coda di risposta.
6. L'aggregatore raccoglie tutti i messaggi di risposta archiviati in un database temporaneo, raggruppa i voli per ID univoco, crea un'unica risposta unificata e la rispedisce al client.

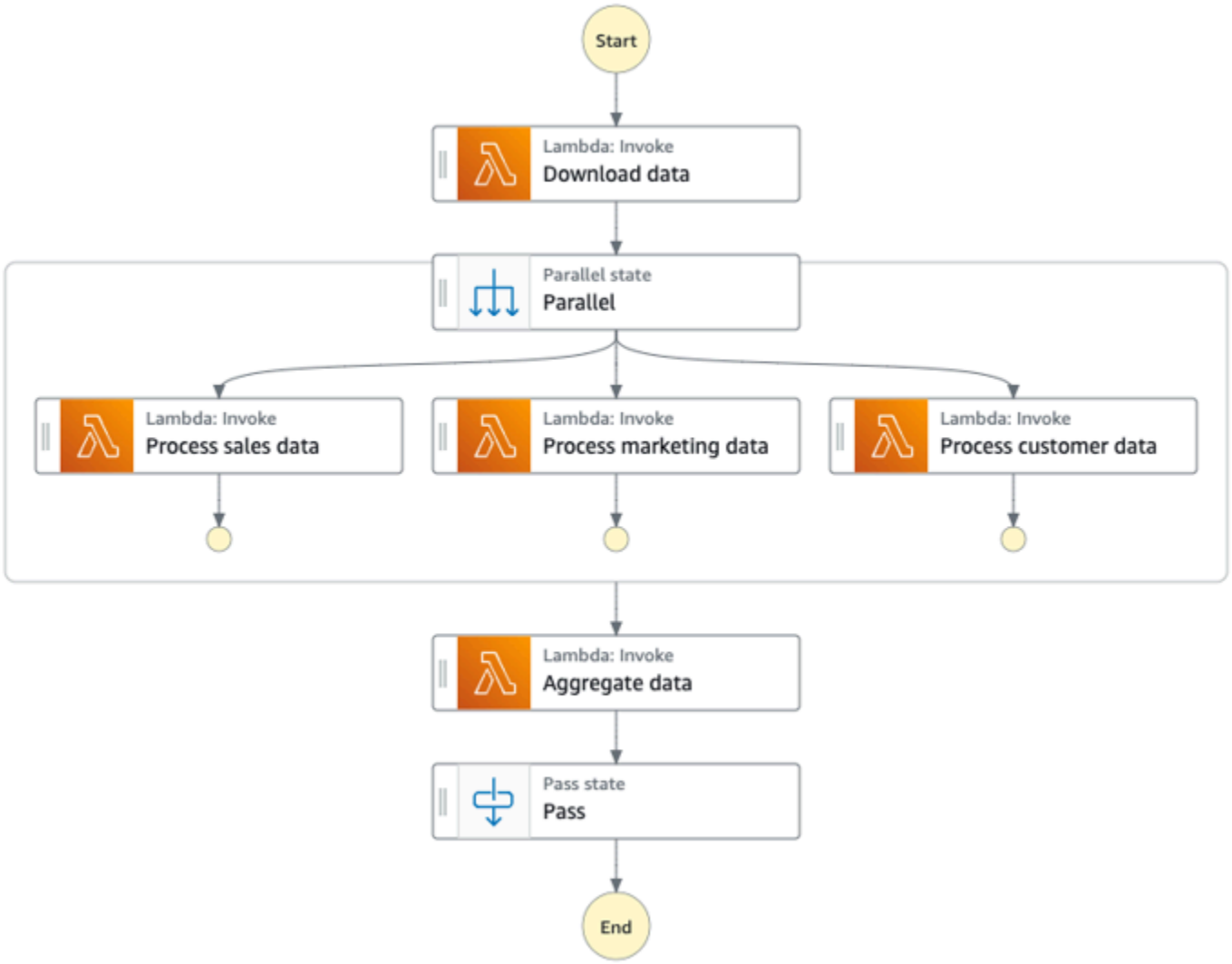
Implementazione utilizzando Servizi AWS

Dispersione per distribuzione

Nella seguente architettura, il root controller è un processore di file di dati (Amazon ECS) che suddivide i dati della richiesta in entrata in singoli bucket Amazon Simple Storage Service (Amazon S3) e avvia un flusso di lavoro. AWS Step Functions Il flusso di lavoro scarica i dati e avvia l'elaborazione parallela dei file. Lo `Parallel` stato attende che tutte le attività restituiscano una risposta. Una AWS Lambda funzione aggrega i dati e li salva su Amazon S3.

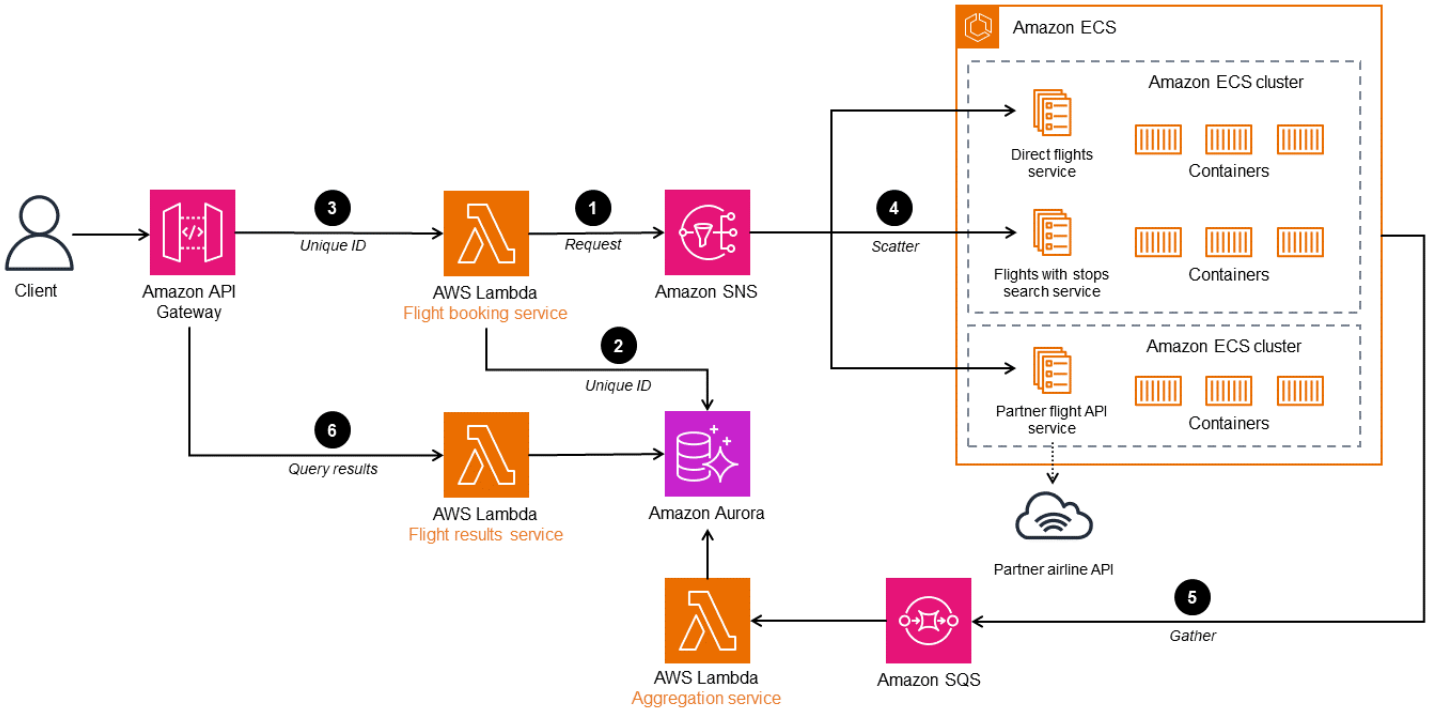


Il diagramma seguente illustra il flusso di lavoro Step Functions con lo `Parallel` stato.



Scatter per asta

Il diagramma seguente mostra un' AWS architettura per il metodo Scatter by auction. Il servizio di prenotazione dei voli root controller distribuisce la richiesta di ricerca del volo su più microservizi. Un canale di pubblicazione e iscrizione è implementato con Amazon Simple Notification Service (Amazon SNS), un servizio di messaggistica gestito per le comunicazioni. Amazon SNS supporta messaggi tra applicazioni di microservizi disaccoppiate o comunicazioni dirette agli utenti. Puoi distribuire i microservizi del destinatario su Amazon Elastic Kubernetes Service (Amazon EKS) o Amazon Elastic Container Service (Amazon ECS) per una migliore gestione e scalabilità. Il servizio per i risultati dei voli restituisce i risultati al cliente. Può essere implementato in AWS Lambda o in altri servizi di orchestrazione di container come Amazon ECS o Amazon EKS.



1. Il servizio di prenotazione dei voli (controller) accetta i criteri di ricerca come input dal cliente, elabora e pubblica la richiesta sull'argomento SNS.
2. Il controller pubblica l'ID univoco in un database Amazon Aurora per identificare la richiesta.
3. Il client invia l'ID univoco al client per la fase 6.
4. I microservizi di ricerca delle prenotazioni che hanno sottoscritto l'argomento di prenotazione ricevono la richiesta.
5. I microservizi elaborano la richiesta e restituiscono la disponibilità dei posti per i criteri di ricerca specificati in una coda di risposta in Amazon Simple Queue Service (Amazon SQS). L'aggregatore raccoglie tutti i messaggi di risposta e li archivia in un database temporaneo.
6. Il servizio di risultati dei voli raggruppa i voli in base a un ID univoco, crea un'unica risposta unificata e la rispedisce al cliente.

Se desideri aggiungere un'altra ricerca di compagnie aeree a questa architettura, aggiungi un microservizio che sottoscrive l'argomento SNS e pubblica nella coda SQS.

Riassumendo, il pattern scatter-collect consente ai sistemi distribuiti di ottenere una parallelizzazione efficiente, ridurre la latenza e gestire senza problemi le comunicazioni asincrone.

GitHub deposito

Per un'implementazione completa dell'architettura di esempio per questo modello, consultate il GitHub repository all'indirizzo https://github.com/aws-samples/ asynchronous-messaging-workshop / tree/master/code/lab-3.

Workshop

- Laboratorio Scatter-collect nel workshop Decoupled Microservices

Riferimenti del blog

- Modelli di integrazione delle applicazioni per microservizi

Contenuti correlati

- Schema di pubblicazione e sottoscrizione

Motivo a fico Strangler

Intento

Lo strangler fig pattern aiuta a migrare un'applicazione monolitica verso un'architettura di microservizi in modo incrementale, con minori rischi di trasformazione e interruzioni aziendali.

Motivazione

Le applicazioni monolitiche sono sviluppate per fornire la maggior parte delle loro funzionalità all'interno di un singolo processo o contenitore. Il codice è strettamente collegato. Di conseguenza, le modifiche alle applicazioni richiedono un nuovo test approfondito per evitare problemi di regressione. Le modifiche non possono essere testate isolatamente, il che influisce sulla durata del ciclo. Poiché l'applicazione è arricchita da più funzionalità, un'elevata complessità può comportare un aumento del tempo dedicato alla manutenzione, un aumento del time-to-market e, di conseguenza, una lenta innovazione del prodotto.

Quando le dimensioni dell'applicazione sono scalabili, aumenta il carico cognitivo del team e può causare confini poco chiari sulla proprietà del team. Non è possibile scalare le singole funzionalità in base al carico: l'intera applicazione deve essere ridimensionata per supportare i picchi di carico. Con l'invecchiamento dei sistemi, la tecnologia può diventare obsoleta, il che fa aumentare i costi di supporto. Le applicazioni monolitiche e preesistenti seguono le best practice disponibili al momento dello sviluppo e non progettate per essere distribuite.

Quando un'applicazione monolitica viene migrata in un'architettura di microservizi, può essere suddivisa in componenti più piccoli. Questi componenti possono essere scalati in modo indipendente, possono essere rilasciati indipendentemente e possono essere di proprietà di singoli team. Ciò si traduce in una maggiore velocità di cambiamento, poiché le modifiche sono localizzate e possono essere testate e rilasciate rapidamente. Le modifiche hanno un ambito di impatto minore perché i componenti sono accoppiati in modo flessibile e possono essere implementati singolarmente.

Sostituire completamente un monolite con un'applicazione di microservizi riscrivendo o rifattorizzando il codice è un'impresa enorme e un grosso rischio. Una migrazione di grande portata, in cui il monolite viene migrato in un'unica operazione, comporta rischi di trasformazione e interruzione delle attività aziendali. Durante il refactoring dell'applicazione, è estremamente difficile o addirittura impossibile aggiungere nuove funzionalità.

Un modo per risolvere questo problema è utilizzare lo Strangler Fig Pattern, introdotto da Martin Fowler. Questo modello prevede il passaggio ai microservizi mediante l'estrazione graduale delle funzionalità e la creazione di una nuova applicazione attorno al sistema esistente. Le funzionalità del monolite vengono gradualmente sostituite dai microservizi e gli utenti delle applicazioni sono in grado di utilizzare progressivamente le nuove funzionalità migrate. Quando tutte le funzionalità vengono trasferite nel nuovo sistema, l'applicazione monolitica può essere disattivata in sicurezza.

Applicabilità

Usa lo schema Strangler Fig quando:

- Desiderate migrare gradualmente la vostra applicazione monolitica verso un'architettura di microservizi.
- Un approccio di migrazione basato sul big bang è rischioso a causa delle dimensioni e della complessità del monolite.
- L'azienda desidera aggiungere nuove funzionalità e non vede l'ora che la trasformazione sia completa.
- Gli utenti finali devono subire un impatto minimo durante la trasformazione.

Problemi e considerazioni

- Accesso alla base di codice: per implementare il pattern strangler fig, è necessario avere accesso alla base di codice dell'applicazione monolith. Man mano che le funzionalità vengono migrate fuori dal monolite, sarà necessario apportare piccole modifiche al codice e implementare un livello anticorruzione all'interno del monolite per indirizzare le chiamate verso nuovi microservizi. Non è possibile intercettare le chiamate senza l'accesso al codice base. L'accesso alla base di codice è fondamentale anche per reindirizzare le richieste in entrata: potrebbe essere necessario rifattorizzare il codice in modo che il livello proxy possa intercettare le chiamate alle funzionalità migrate e indirizzarle verso i microservizi.
- Dominio poco chiaro: la scomposizione prematura dei sistemi può essere costosa, soprattutto quando il dominio non è chiaro ed è possibile che i limiti del servizio siano errati. La progettazione basata sul dominio (DDD) è un meccanismo per comprendere il dominio e l'event storming è una tecnica per determinare i confini del dominio.
- Identificazione dei microservizi: è possibile utilizzare DDD come strumento chiave per identificare i microservizi. Per identificare i microservizi, cerca le divisioni naturali tra le classi di servizio. Molti

servizi avranno il proprio oggetto di accesso ai dati e si disaccoppieranno facilmente. I servizi che hanno una logica aziendale correlata e classi che non hanno alcuna dipendenza o hanno poche dipendenze sono buoni candidati per i microservizi. È possibile rifattorizzare il codice prima di scomporre il monolite per evitare un accoppiamento stretto. Dovresti inoltre considerare i requisiti di conformità, la cadenza di rilascio, la posizione geografica dei team, le esigenze di scalabilità, le esigenze tecnologiche basate sui casi d'uso e il carico cognitivo dei team.

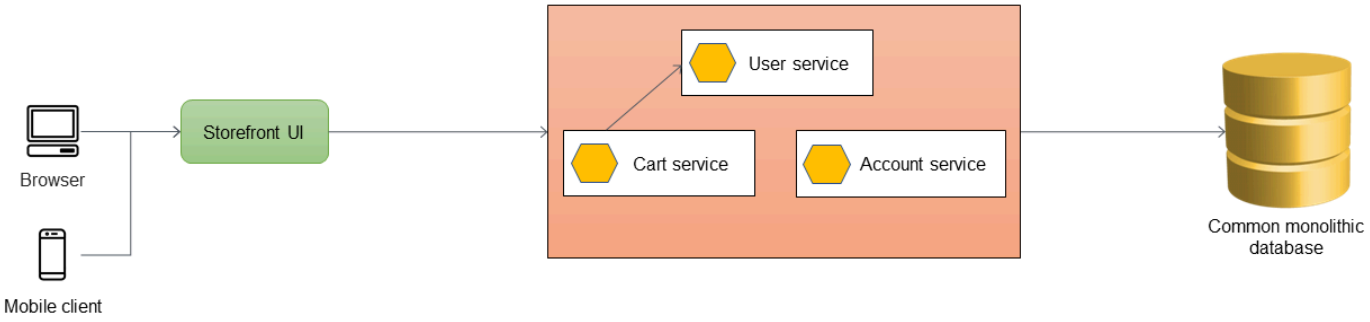
- Livello anticorruzione: durante il processo di migrazione, quando le funzionalità all'interno del monolite devono richiamare le funzionalità che sono state migrate come microservizi, è necessario implementare un livello anticorruzione (ACL) che indirizzi ogni chiamata al microservizio appropriato. Per disaccoppiare e impedire la modifica dei chiamanti esistenti all'interno del monolite, l'ACL funge da adattatore o da facciata che converte le chiamate nella nuova interfaccia. Questo è discusso in dettaglio nella sezione Implementazione del modello ACL riportata in precedenza in questa guida.
- Errore a livello proxy: durante la migrazione, un livello proxy intercetta le richieste che arrivano all'applicazione monolitica e le indirizza al sistema legacy o al nuovo sistema. Tuttavia, questo livello proxy può diventare un singolo punto di errore o un ostacolo alle prestazioni.
- Complessità dell'applicazione: i monoliti di grandi dimensioni traggono il massimo vantaggio dal pattern Strangler Fig. Per le applicazioni di piccole dimensioni, in cui la complessità del refactoring completo è bassa, potrebbe essere più efficiente riscrivere l'applicazione in un'architettura di microservizi anziché migrarla.
- Interazioni di servizio: i microservizi possono comunicare in modo sincrono o asincrono. Quando è richiesta una comunicazione sincrona, valuta se i timeout possono causare il consumo della connessione o del pool di thread, con conseguenti problemi di prestazioni delle applicazioni. In questi casi, utilizzate lo schema degli interruttori automatici per restituire un guasto immediato in caso di operazioni che rischiano di fallire per lunghi periodi di tempo. La comunicazione asincrona può essere ottenuta utilizzando eventi e code di messaggistica.
- Aggregazione dei dati: in un'architettura di microservizi, i dati vengono distribuiti tra database. Quando è richiesta l'aggregazione dei dati, è possibile utilizzare AWS AppSyncnel front-end o il pattern CQRS (Command Query Responsibility Segregation) nel backend.
- Coerenza dei dati: i microservizi possiedono il proprio archivio dati e anche l'applicazione monolitica può potenzialmente utilizzare questi dati. Per abilitare la condivisione, è possibile sincronizzare l'archivio dati dei nuovi microservizi con il database dell'applicazione monolitica utilizzando una coda e un agente. Tuttavia, ciò può causare la ridondanza dei dati e l'eventuale coerenza tra due archivi di dati, quindi consigliamo di considerarla una soluzione tattica fino a quando non sarà possibile stabilire una soluzione a lungo termine come un data lake.

Implementazione

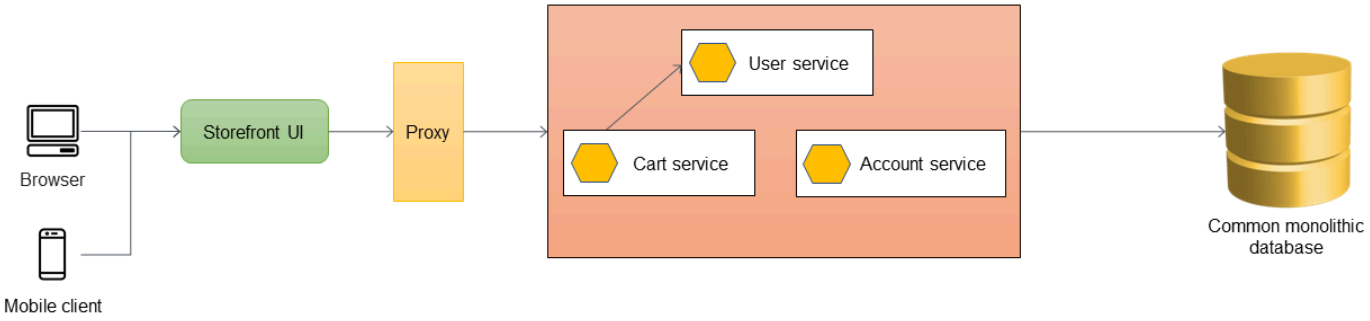
Nel modello Strangler Fig, si sostituiscono funzionalità specifiche con un nuovo servizio o applicazione, un componente alla volta. Un livello proxy intercetta le richieste che arrivano all'applicazione monolitica e le indirizza al sistema precedente o al nuovo sistema. Poiché il livello proxy indirizza gli utenti all'applicazione corretta, è possibile aggiungere funzionalità al nuovo sistema assicurando al contempo che il monolite continui a funzionare. Il nuovo sistema alla fine sostituisce tutte le funzionalità del vecchio sistema ed è possibile disattivarlo.

Architettura di alto livello

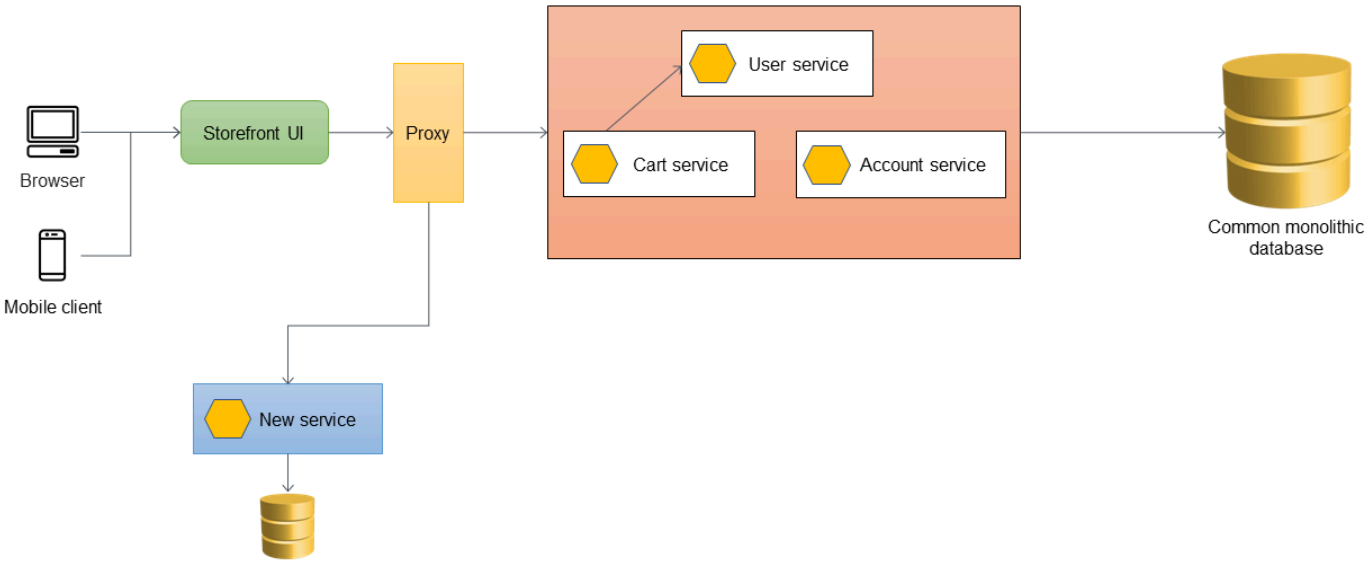
Nel diagramma seguente, un'applicazione monolitica dispone di tre servizi: servizio utente, servizio carrello e servizio account. Il servizio cart dipende dal servizio utente e l'applicazione utilizza un database relazionale monolitico.



Il primo passaggio consiste nell'aggiungere un livello proxy tra l'interfaccia utente dello storefront e l'applicazione monolitica. All'inizio, il proxy indirizza tutto il traffico verso l'applicazione monolitica.

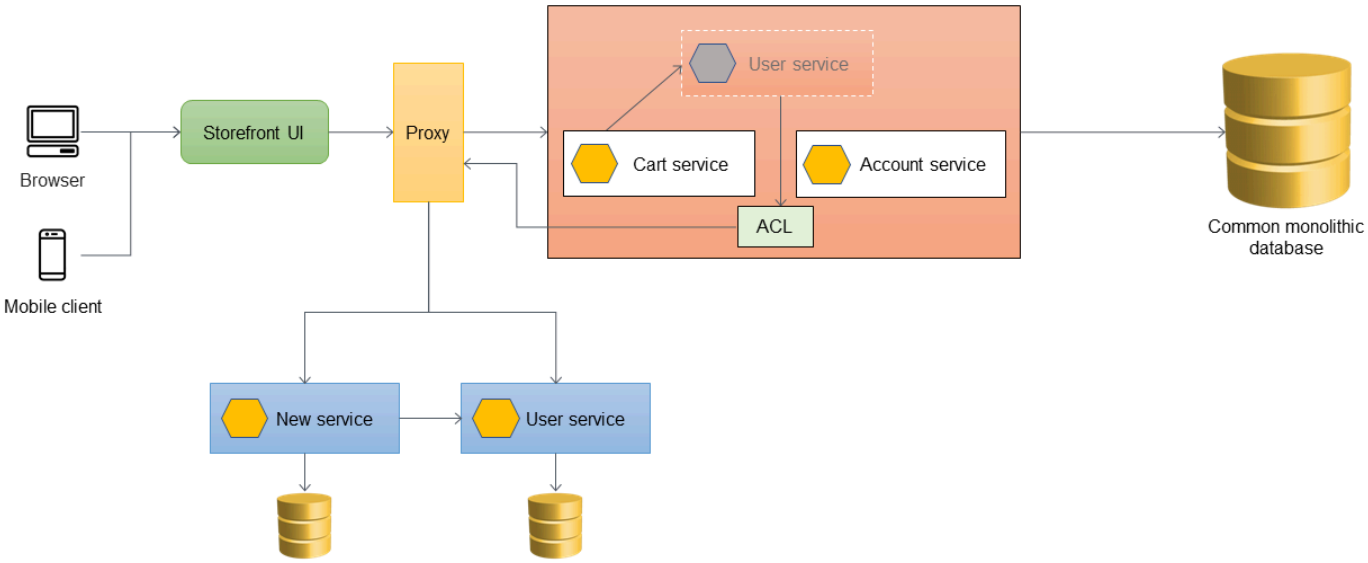


Quando desideri aggiungere nuove funzionalità alla tua applicazione, le implementi come nuovi microservizi invece di aggiungere funzionalità al monolite esistente. Tuttavia, continuate a correggere i bug nel monolite per garantire la stabilità dell'applicazione. Nel diagramma seguente, il livello proxy indirizza le chiamate al monolite o al nuovo microservizio in base all'URL dell'API.



Aggiungere un livello anticorruzione

Nella seguente architettura, il servizio utente è stato migrato a un microservizio. Il servizio cart richiama il servizio utente, ma l'implementazione non è più disponibile all'interno del monolite. Inoltre, l'interfaccia del servizio appena migrato potrebbe non corrispondere all'interfaccia precedente all'interno dell'applicazione monolitica. Per risolvere queste modifiche, è necessario implementare un ACL. Durante il processo di migrazione, quando le funzionalità all'interno del monolite devono richiamare le funzionalità che sono state migrate come microservizi, l'ACL converte le chiamate nella nuova interfaccia e le indirizza al microservizio appropriato.

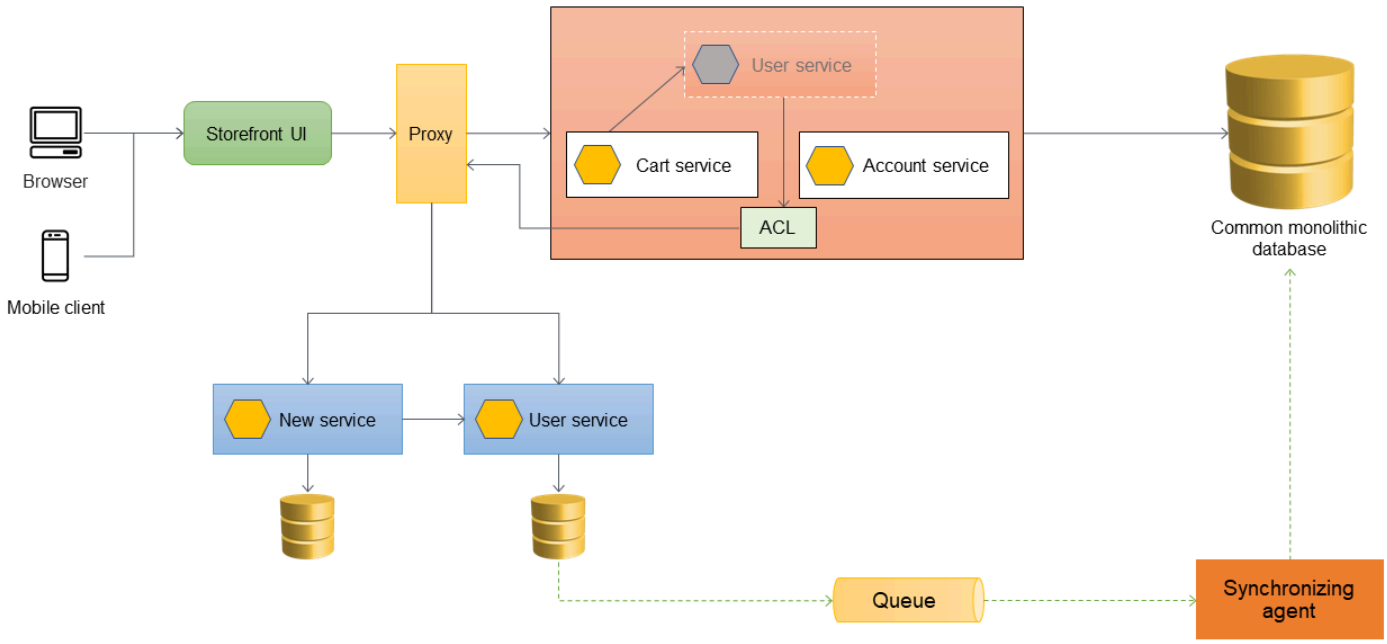


È possibile implementare l'ACL all'interno dell'applicazione monolitica come classe specifica per il servizio che è stato migrato; ad esempio, o. `UserServiceFacade` `UserServiceAdapter` L'ACL deve essere disattivato dopo che tutti i servizi dipendenti sono stati migrati nell'architettura dei microservizi.

Quando si utilizza l'ACL, il servizio cart chiama ancora il servizio utente all'interno del monolite e il servizio utente reindirizza la chiamata al microservizio tramite l'ACL. Il servizio cart dovrebbe comunque chiamare il servizio utente senza essere a conoscenza della migrazione dei microservizi. Questo accoppiamento libero è necessario per ridurre la regressione e le interruzioni dell'attività.

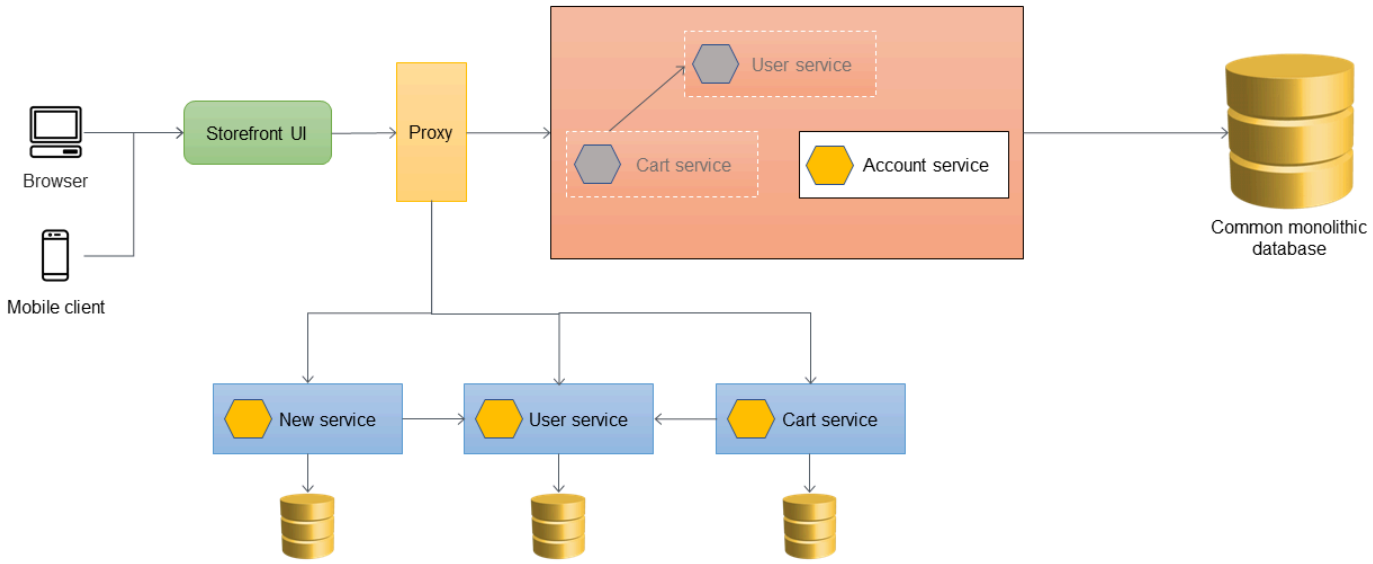
Gestione della sincronizzazione dei dati

Come best practice, il microservizio dovrebbe essere proprietario dei propri dati. Il servizio utente archivia i propri dati nel proprio archivio dati. Potrebbe essere necessario sincronizzare i dati con il database monolitico per gestire dipendenze come la reportistica e supportare applicazioni downstream che non sono ancora pronte per accedere direttamente ai microservizi. L'applicazione monolitica potrebbe inoltre richiedere i dati per altre funzioni e componenti che non sono ancora stati migrati ai microservizi. È quindi necessaria la sincronizzazione dei dati tra il nuovo microservizio e il monolite. Per sincronizzare i dati, è possibile introdurre un agente di sincronizzazione tra il microservizio utente e il database monolitico, come illustrato nel diagramma seguente. Il microservizio utente invia un evento alla coda ogni volta che il relativo database viene aggiornato. L'agente di sincronizzazione ascolta la coda e aggiorna continuamente il database monolitico. I dati nel database monolitico alla fine sono coerenti con i dati che vengono sincronizzati.

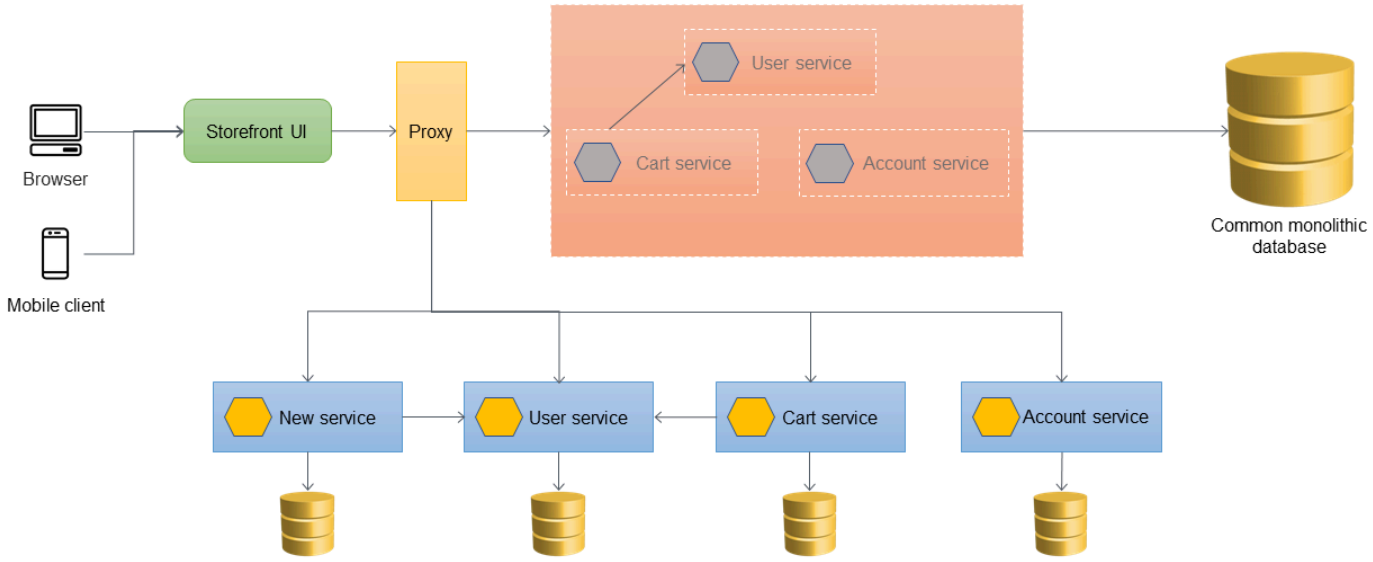


Migrazione di servizi aggiuntivi

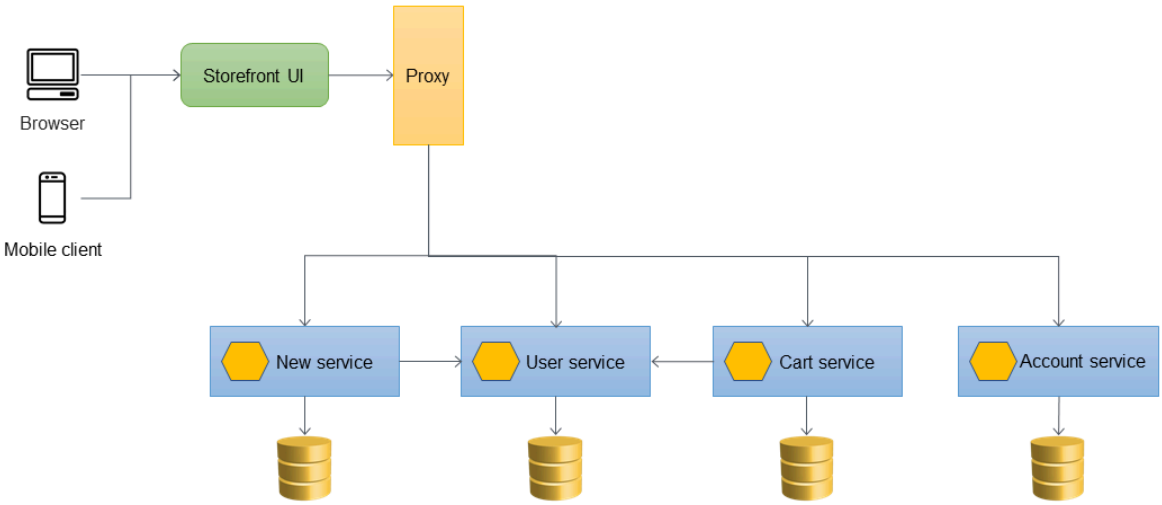
Quando il servizio cart viene migrato dall'applicazione monolitica, il relativo codice viene modificato per richiamare direttamente il nuovo servizio, in modo che l'ACL non instrada più tali chiamate. Il diagramma seguente illustra tale architettura.



Il diagramma seguente mostra lo stato di strangolamento finale in cui tutti i servizi sono stati migrati dal monolite e rimane solo lo scheletro del monolite. I dati storici possono essere migrati in archivi dati di proprietà di singoli servizi. L'ACL può essere rimosso e il monolite è pronto per essere smantellato in questa fase.



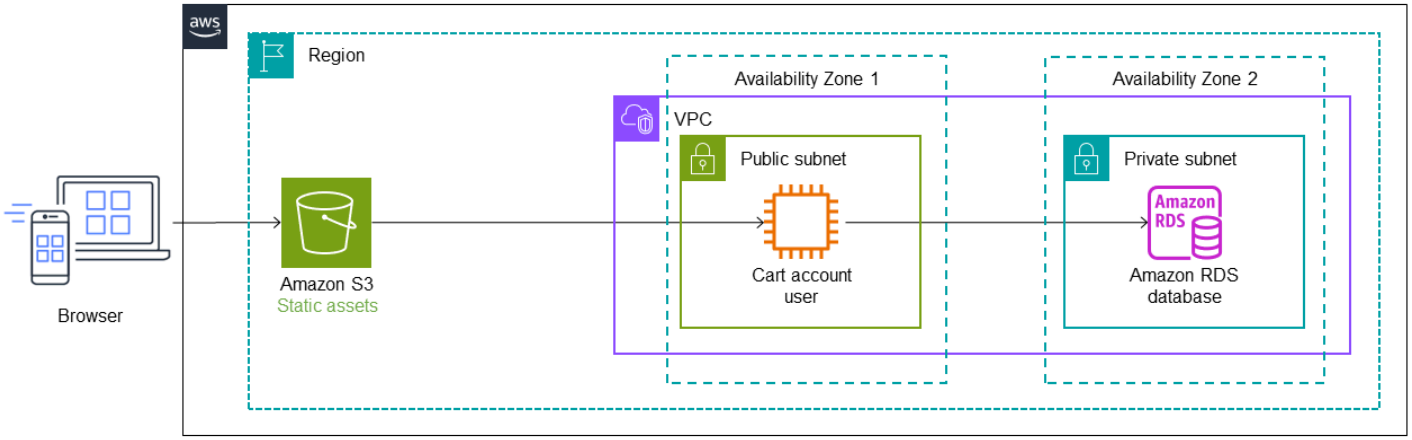
Il diagramma seguente mostra l'architettura finale dopo la disattivazione dell'applicazione monolitica. È possibile ospitare i singoli microservizi tramite un URL basato sulle risorse (ad esempio`http://www.storefront.com/user`) o tramite il proprio dominio (ad esempio) in base ai requisiti dell'applicazione. `http://user.storefront.com` Per ulteriori informazioni sui principali metodi per esporre le API HTTP ai consumatori upstream utilizzando nomi host e percorsi, consulta la sezione sui modelli di routing delle API.



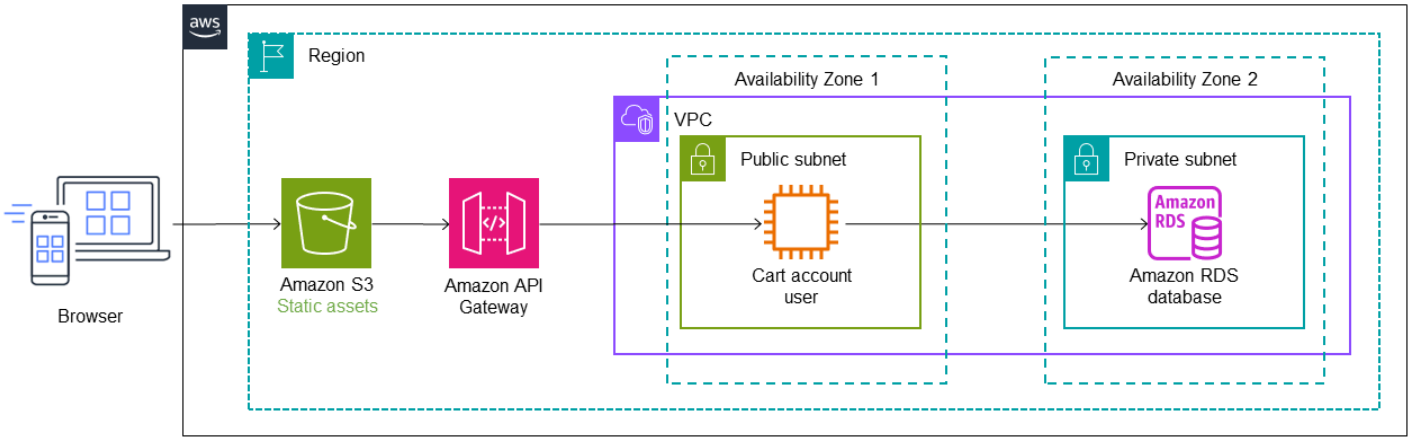
AWSImplementazione tramite servizi

Utilizzo di API Gateway come proxy dell'applicazione

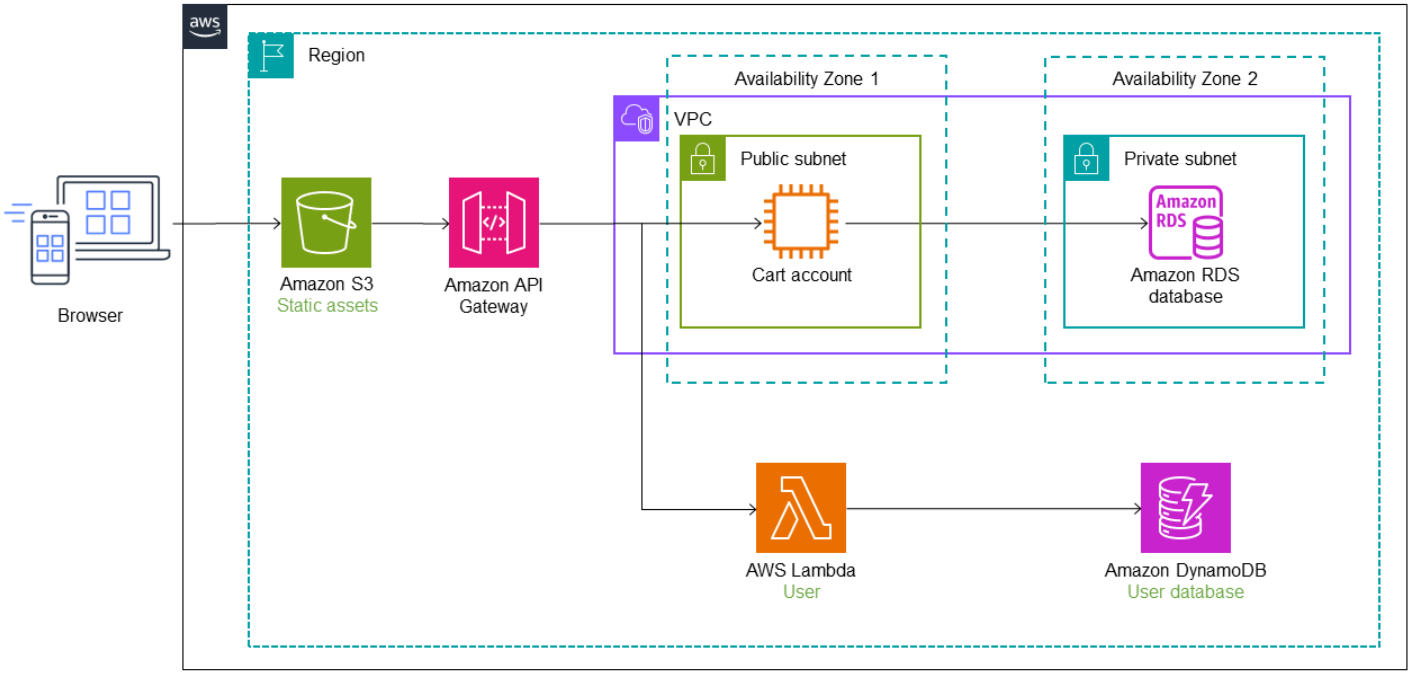
Il diagramma seguente mostra lo stato iniziale dell'applicazione monolitica. Supponiamo che sia stata effettuata la migrazione AWS utilizzando una lift-and-shift strategia, quindi è in esecuzione su un'istanza Amazon Elastic Compute Cloud (Amazon EC2) e utilizzi un database Amazon Relational Database Service (Amazon RDS). Per semplicità, l'architettura utilizza un singolo cloud privato virtuale (VPC) con una sottorete privata e una pubblica e supponiamo che i microservizi vengano inizialmente implementati all'interno della stessa. Account AWS (La migliore pratica negli ambienti di produzione consiste nell'utilizzare un'architettura multi-account per garantire l'indipendenza dell'implementazione.) L'istanza EC2 risiede in una singola zona di disponibilità nella sottorete pubblica e l'istanza RDS risiede in una singola zona di disponibilità nella sottorete privata. Amazon Simple Storage Service (Amazon S3) archivia risorse statiche come file CSS e React per JavaScript il sito Web.



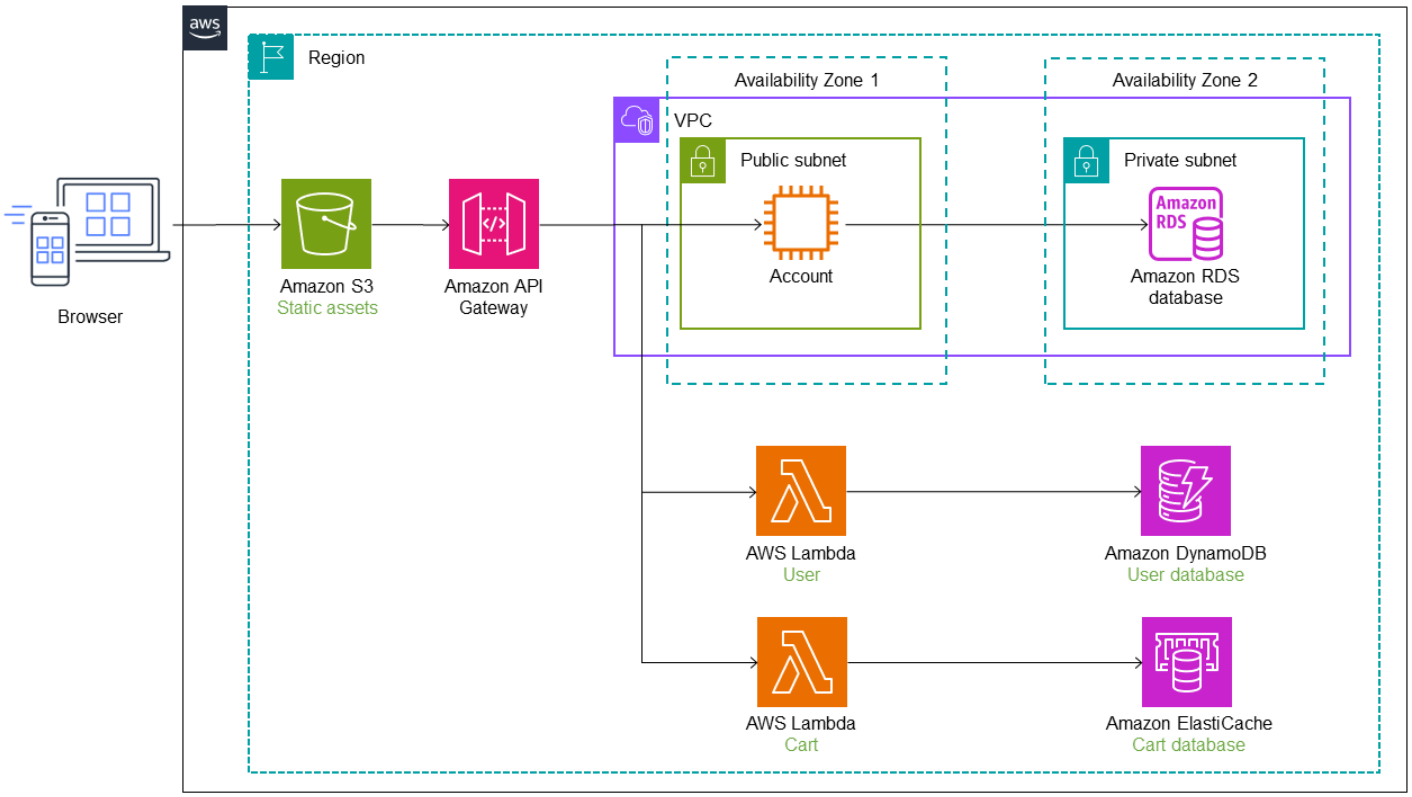
Nella seguente architettura, AWS Migration Hub Refactor Spacesdistribuisce Amazon API Gateway davanti all'applicazione monolitica. Refactor Spaces crea un'infrastruttura di refactoring all'interno del tuo account e API Gateway funge da livello proxy per l'instradamento delle chiamate verso il monolite. Inizialmente, tutte le chiamate vengono instradate all'applicazione monolitica attraverso il livello proxy. Come discusso in precedenza, i livelli proxy possono diventare un unico punto di errore. Tuttavia, l'utilizzo di API Gateway come proxy riduce il rischio perché si tratta di un servizio Multi-AZ senza server.



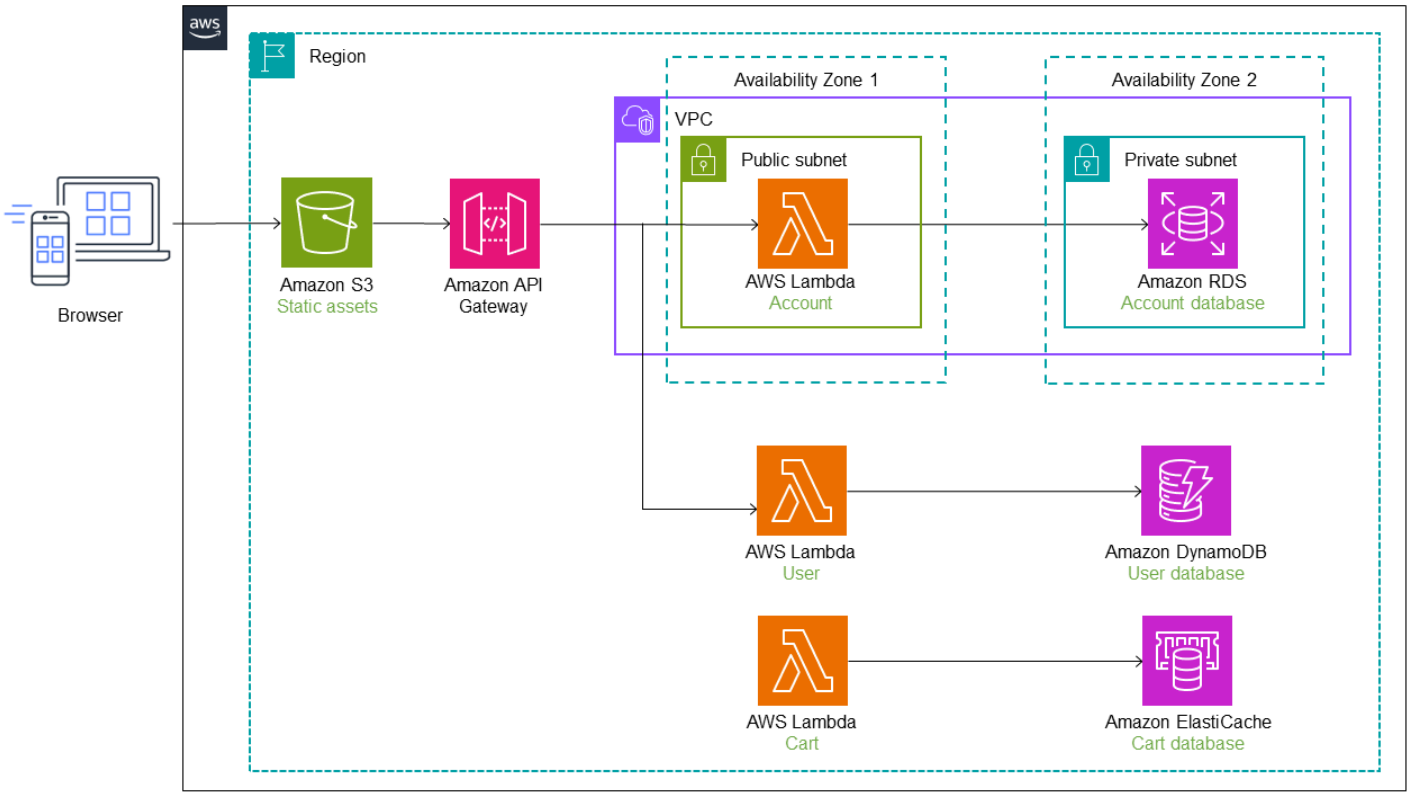
Il servizio utente viene migrato in una funzione Lambda e un database Amazon DynamoDB ne archivia i dati. Un endpoint del servizio Lambda e una route predefinita vengono aggiunti a Refactor Spaces e l'API Gateway viene configurato automaticamente per instradare le chiamate alla funzione Lambda. Per i dettagli sull'implementazione, consulta il Modulo 2 dell'Iterative App Modernization Workshop.



Nel diagramma seguente, anche il servizio cart è stato migrato dal monolite a una funzione Lambda. Un ulteriore endpoint di percorso e servizio viene aggiunto a Refactor Spaces e il traffico passa automaticamente alla funzione Lambda`Cart`. L'archivio dati per la funzione Lambda è gestito da Amazon. ElastiCache L'applicazione monolitica rimane ancora nell'istanza EC2 insieme al database Amazon RDS.



Nel diagramma seguente, l'ultimo servizio (account) viene migrato dal monolite a una funzione Lambda. Continua a utilizzare il database Amazon RDS originale. La nuova architettura ora dispone di tre microservizi con database separati. Ogni servizio utilizza un tipo diverso di database. Questo concetto di utilizzo di database creati appositamente per soddisfare le esigenze specifiche dei microservizi si chiama persistenza poliglotta. Le funzioni Lambda possono essere implementate anche in diversi linguaggi di programmazione, a seconda del caso d'uso. Durante il refactoring, Refactor Spaces automatizza il cutover e l'instradamento del traffico verso Lambda. Ciò consente ai costruttori di risparmiare il tempo necessario per progettare, implementare e configurare l'infrastruttura di routing.



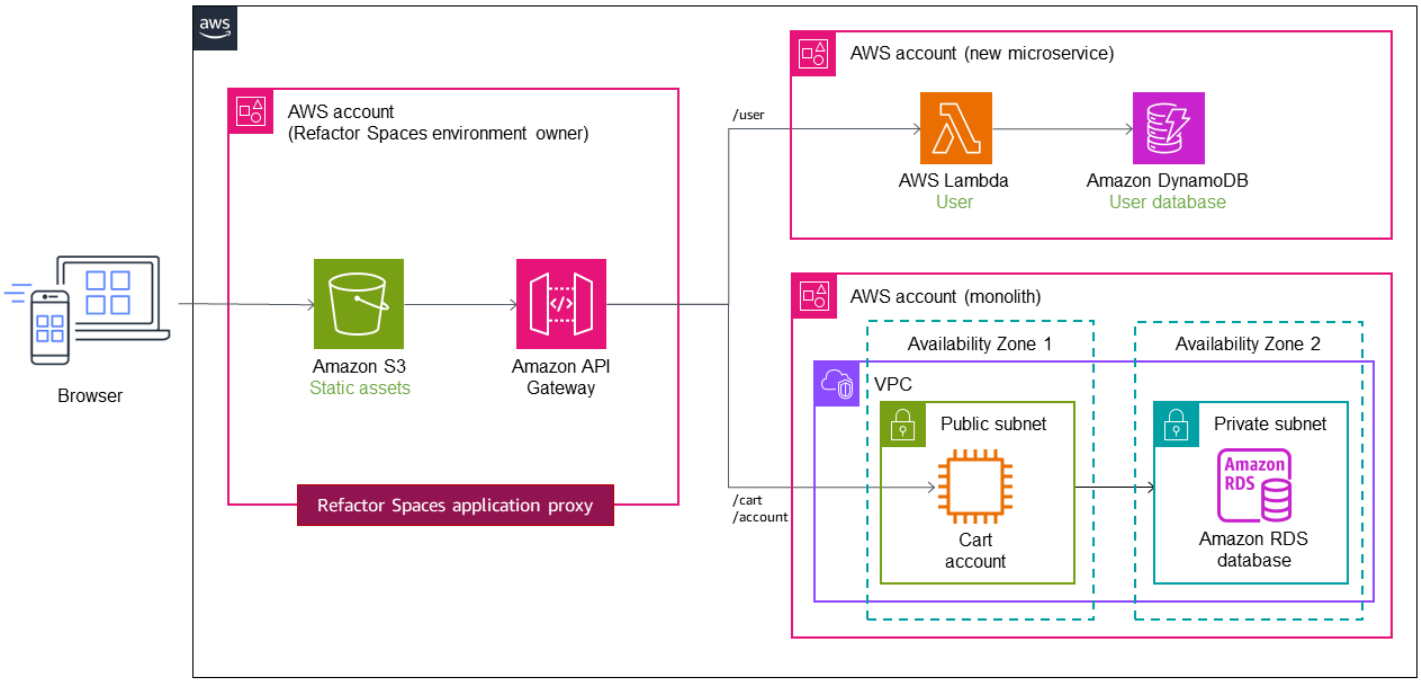
Utilizzo di più account

Nell'implementazione precedente, utilizzavamo un singolo VPC con una sottorete privata e una pubblica per l'applicazione monolitica e implementavamo i microservizi all'interno della stessa per motivi di semplicità. Account AWS Tuttavia, ciò si verifica raramente negli scenari del mondo reale, in cui i microservizi vengono spesso distribuiti in più formati per garantire l'indipendenza dall'implementazione. Account AWS In una struttura con più account, è necessario configurare il routing del traffico dal monolite ai nuovi servizi in diversi account.

Refactor Spaces ti aiuta a creare e configurare l'AWSinfrastruttura per il routing delle chiamate API lontano dall'applicazione monolitica. Refactor Spaces orchestra API Gateway, Network Load Balancer e policy AWS Identity and Access Managementbasate sulle risorse (IAM) all'interno dei AWS tuoi account come parte della sua risorsa applicativa. Puoi aggiungere in modo trasparente nuovi servizi in uno Account AWS o più account a un endpoint HTTP esterno. Tutte queste risorse sono orchestrate all'interno dell'azienda Account AWS e possono essere personalizzate e configurate dopo l'implementazione.

Supponiamo che i servizi user e cart siano distribuiti su due account diversi, come illustrato nel diagramma seguente. Quando utilizzi Refactor Spaces, devi solo configurare l'endpoint del servizio e

il percorso. Refactor Spaces automatizza l'integrazione API Gateway-Lambda e la creazione di policy sulle risorse Lambda, in modo che tu possa concentrarti sul refactoring sicuro dei servizi fuori dal monolite.



Per un video tutorial sull'utilizzo di Refactor Spaces, vedi Refactor Apps Incrementally with. AWS Migration Hub Refactor Spaces

Workshop

- Workshop iterativo sulla modernizzazione delle app

Riferimenti del blog

- AWS Migration Hub Refactor Spaces
- Approfondisci su un AWS Migration Hub Refactor Spaces
- Deployment Pipeline, architettura di riferimento e implementazioni di riferimento

Contenuti correlati

- Modelli di routing delle API

- Documentazione Refactor Spaces

Modello transazionale di posta in uscita

Intento

Il modello transazionale di posta in uscita risolve il problema delle operazioni di doppia scrittura che si verifica nei sistemi distribuiti quando una singola operazione comporta sia un'operazione di scrittura del database che una notifica di messaggio o evento. Un'operazione di doppia scrittura si verifica quando un'applicazione scrive su due sistemi diversi; ad esempio, quando un microservizio deve mantenere i dati nel database e inviare un messaggio per notificare altri sistemi. Un errore in una di queste operazioni potrebbe causare dati non coerenti.

Motivazione

Quando un microservizio invia una notifica di evento dopo un aggiornamento del database, queste due operazioni devono essere eseguite in modo atomico per garantire la coerenza e l'affidabilità dei dati.

- Se l'aggiornamento del database ha esito positivo ma la notifica dell'evento non riesce, il servizio a valle non verrà a conoscenza della modifica e il sistema potrebbe entrare in uno stato incoerente.
- Se l'aggiornamento del database non riesce ma viene inviata la notifica dell'evento, i dati potrebbero danneggiarsi, il che potrebbe influire sull'affidabilità del sistema.

Applicabilità

Utilizza il modello di posta in uscita transazionale quando:

- Stai creando un'applicazione basata sugli eventi in cui un aggiornamento del database avvia una notifica di evento.
- Vuoi garantire l'atomicità nelle operazioni che coinvolgono due servizi.
- Desideri implementare il modello di approvvigionamento degli eventi.

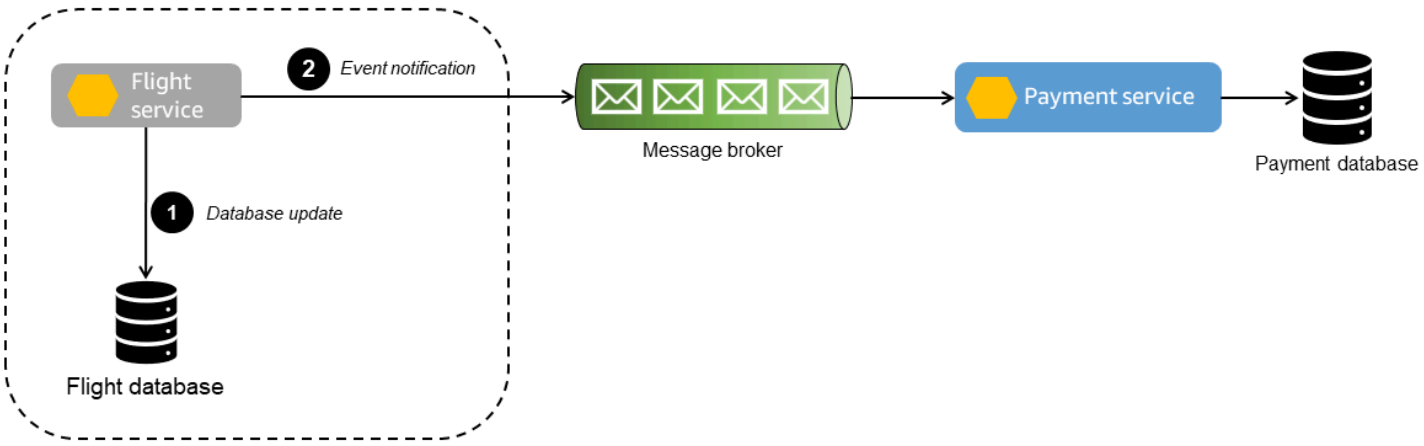
Problemi e considerazioni

- Messaggi duplicati: il servizio di elaborazione degli eventi potrebbe inviare messaggi o eventi duplicati, pertanto ti consigliamo di rendere idempotente il servizio di consumo monitorando i messaggi elaborati.
- Ordine di notifica: invia messaggi o eventi nello stesso ordine in cui il servizio aggiorna il database. Questo è fondamentale per il modello di approvvigionamento degli eventi in cui è possibile utilizzare un archivio eventi per il point-in-time ripristino del data store. Se l'ordine non è corretto, potrebbe compromettere la qualità dei dati. L'eventuale coerenza e il rollback del database possono aggravare il problema se l'ordine delle notifiche non viene mantenuto.
- Ripristino delle transazioni: non inviare una notifica di evento se la transazione viene ripristinata.
- Gestione delle transazioni a livello di servizio: se la transazione include servizi che richiedono aggiornamenti dei datastore, utilizza il modello di orchestrazione Saga per preservare l'integrità dei dati nei datastore.

Implementazione

Architettura di alto livello

Il seguente diagramma di sequenza mostra l'ordine degli eventi che si verificano durante le operazioni di doppia scrittura.



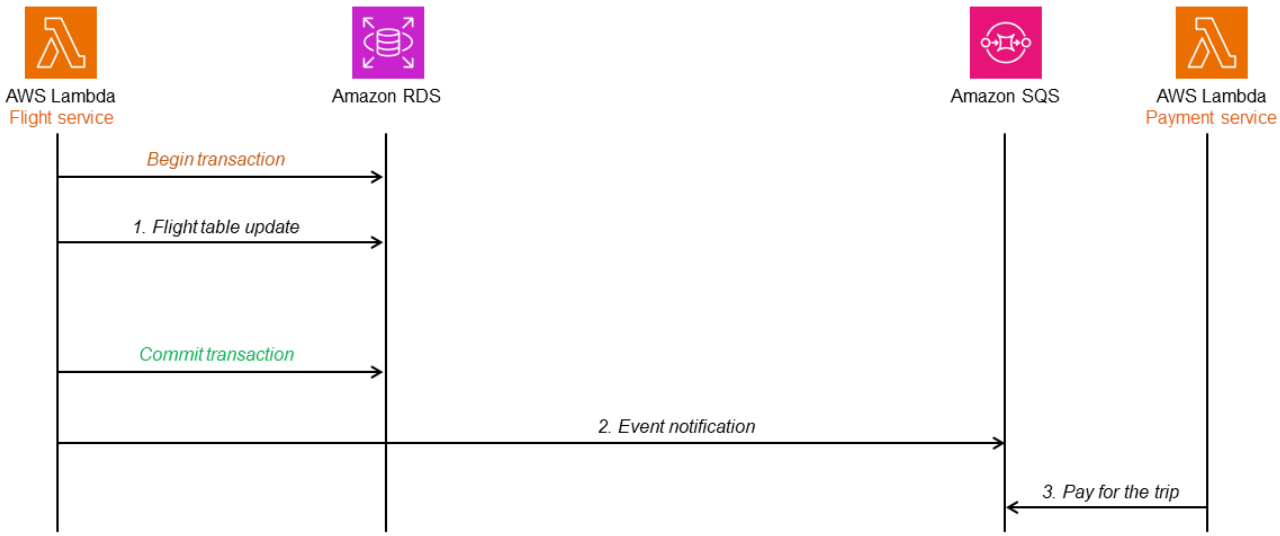
1. Il servizio in corso scrive nel database e invia una notifica di evento al servizio di pagamento.
2. Il broker di messaggi trasmette i messaggi e gli eventi al servizio di pagamento. Qualsiasi errore nel broker di messaggi impedisce al servizio di pagamento di ricevere gli aggiornamenti.

Se l'aggiornamento del database in corso non riesce ma la notifica viene inviata, il servizio di pagamento elaborerà il pagamento in base alla notifica dell'evento. Ciò causerà incongruenze nei dati a valle.

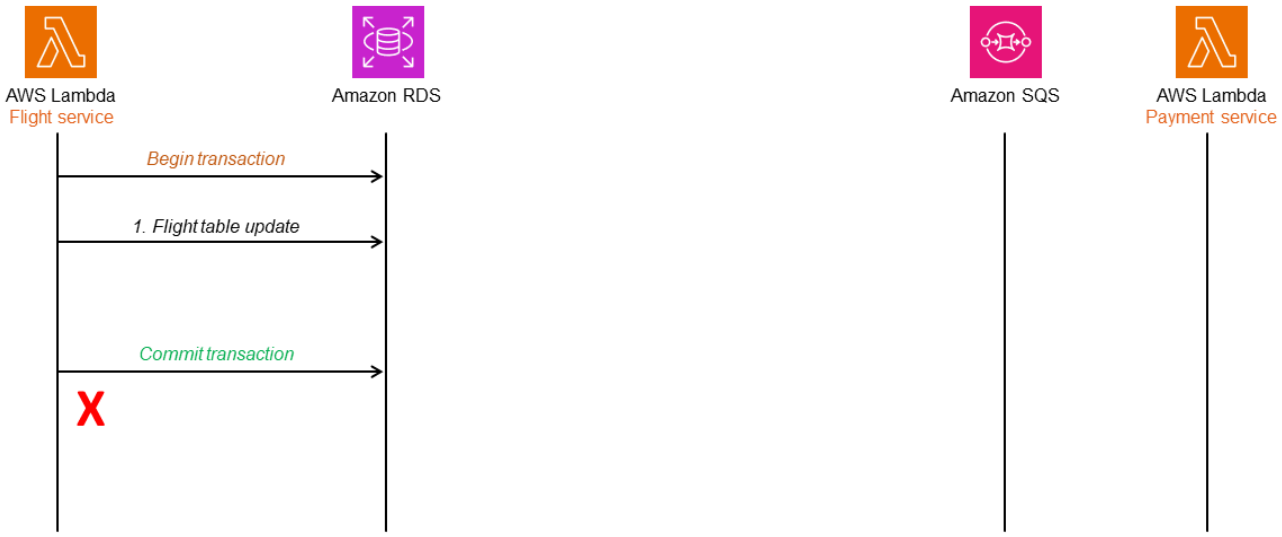
Implementazione tramite servizi AWS

Per illustrare lo schema nel diagramma di sequenza, utilizzeremo i seguenti AWS servizi, come illustrato nel diagramma seguente.

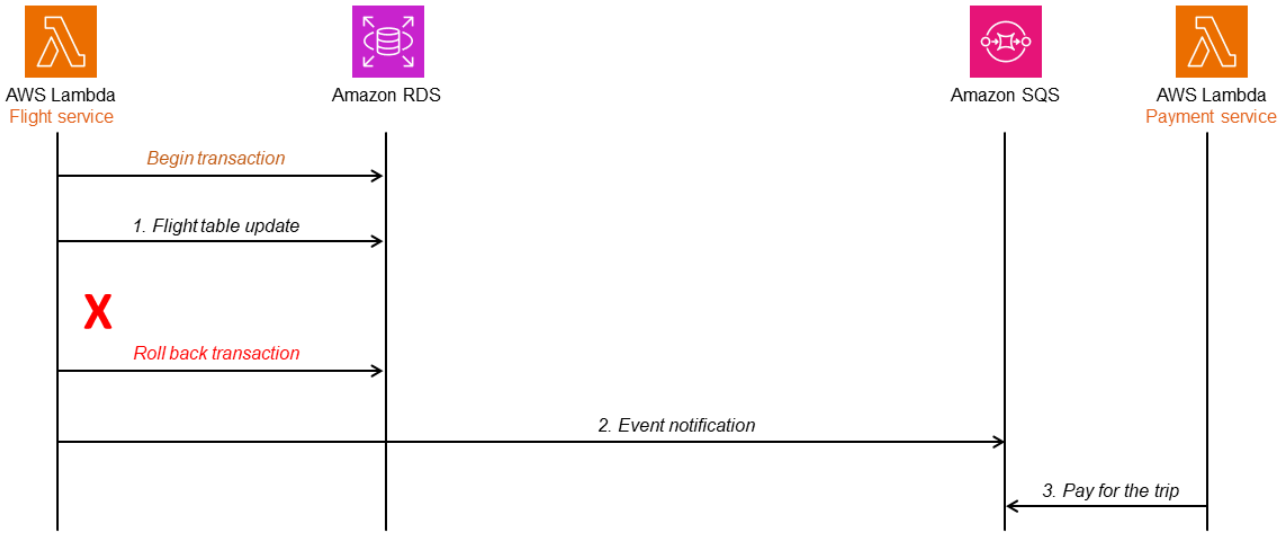
- I microservizi vengono implementati utilizzando AWS Lambda.
- Il database primario è gestito da Amazon Relational Database Service (Amazon RDS).
- Amazon Simple Queue Service (Amazon SQS) funge da broker di messaggi che riceve le notifiche degli eventi.



Se il servizio in corso fallisce dopo aver effettuato la transazione, ciò potrebbe comportare il mancato invio della notifica dell'evento.



Tuttavia, la transazione potrebbe fallire e tornare indietro, ma la notifica dell'evento potrebbe comunque essere inviata, causando l'elaborazione del pagamento da parte del servizio di pagamento.



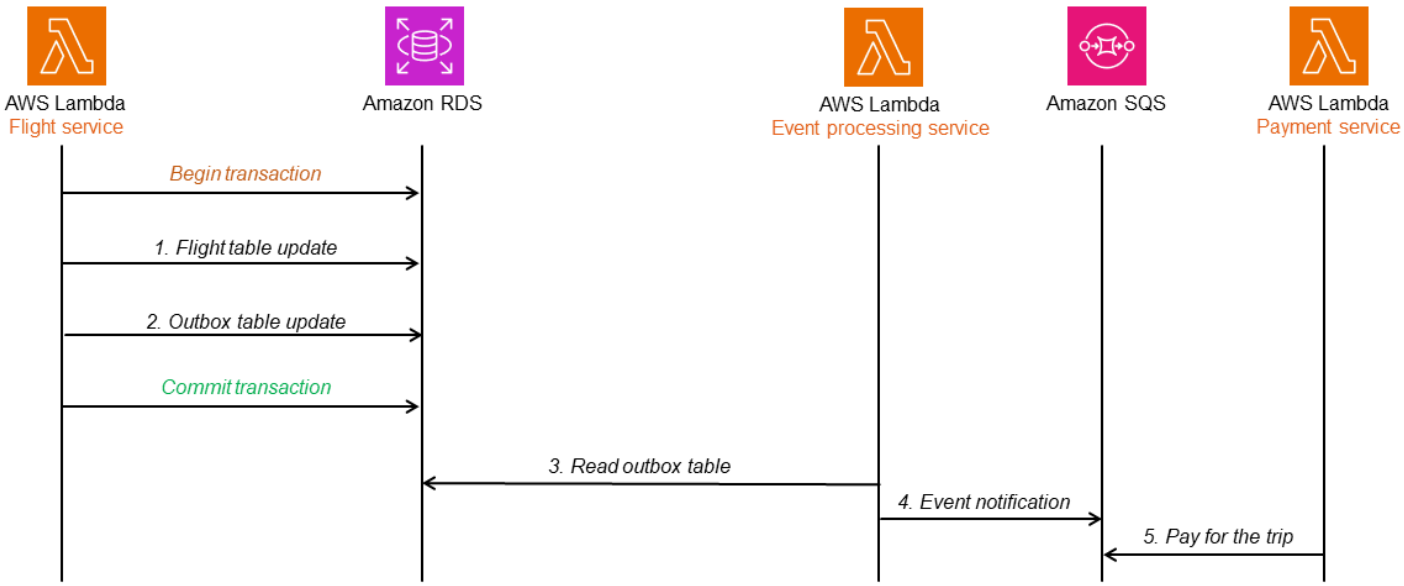
Per risolvere questo problema, puoi utilizzare una tabella di posta in uscita o la funzionalità CDC (Change Data Capture). Le sezioni seguenti illustrano queste due opzioni e come implementarle utilizzando i servizi AWS.

Utilizzo di una tabella di posta in uscita con un database relazionale

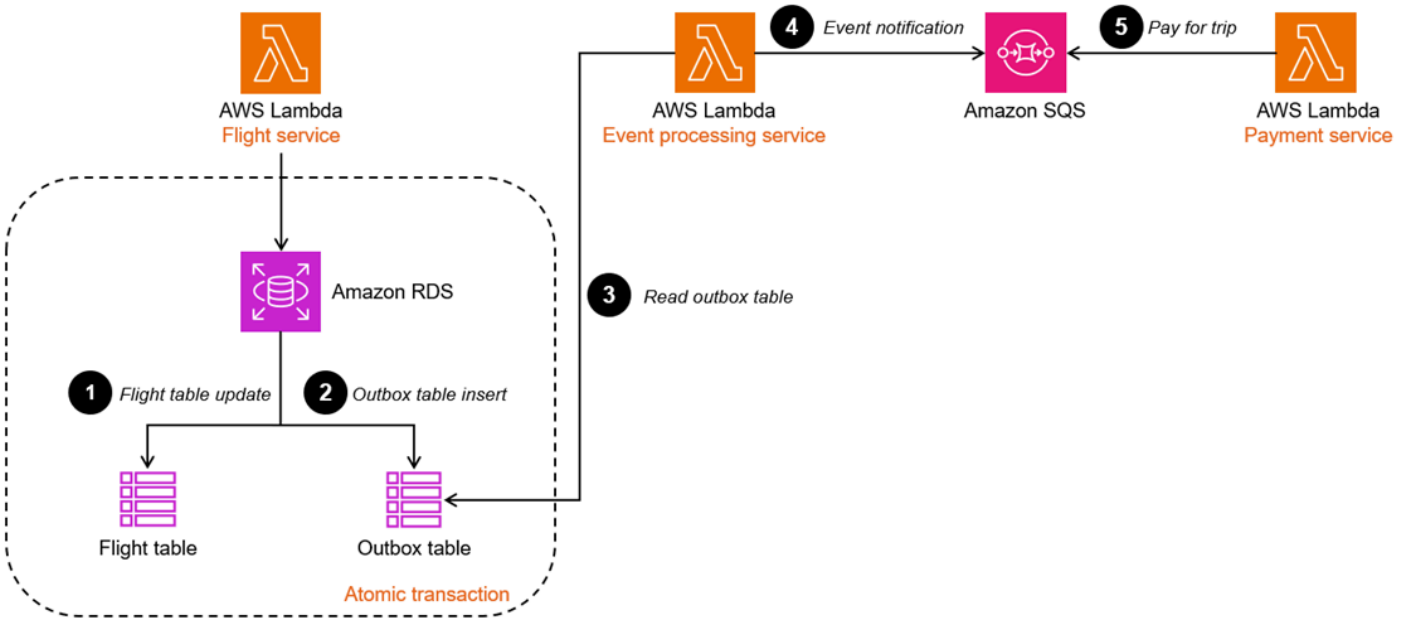
Una tabella di posta in uscita memorizza tutti gli eventi del servizio in corso con un timestamp e un numero di sequenza.

Quando la tabella in corso viene aggiornata, anche la tabella di posta in uscita viene aggiornata nella stessa transazione. Un altro servizio (ad esempio, il servizio di elaborazione degli eventi) legge dalla tabella di posta in uscita e invia l'evento ad Amazon SQS. Amazon SQS invia un messaggio sull'evento al servizio di pagamento per un'ulteriore elaborazione. Le code standard di Amazon SQS garantiscono che il messaggio venga recapitato almeno una volta e non vada perso. Tuttavia, quando utilizzi le code standard di Amazon SQS, lo stesso messaggio o evento potrebbe essere recapitato più di una volta, quindi dovresti assicurarti che il servizio di notifica degli eventi sia idempotente (ovvero che l'elaborazione dello stesso messaggio più volte non abbia effetti negativi). Se desideri che il messaggio venga recapitato esattamente una sola volta, con l'ordinamento dei messaggi, puoi utilizzare le code FIFO (first in, first out) di Amazon SQS.

Se l'aggiornamento della tabella in corso non riesce o l'aggiornamento della tabella di posta in uscita non riesce, l'intera transazione viene ripristinata, quindi non ci sono incongruenze nei dati a valle.



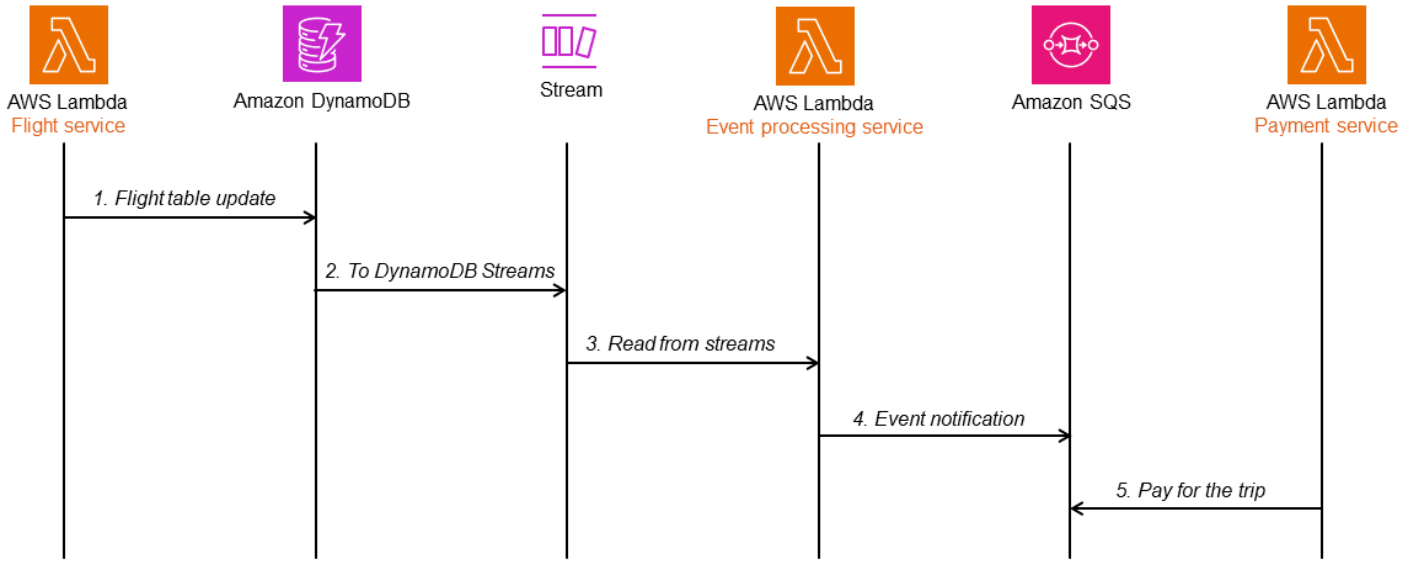
Nel diagramma seguente, l'architettura transazionale della posta in uscita viene implementata utilizzando un database Amazon RDS. Quando il servizio di elaborazione degli eventi legge la tabella dei messaggi in uscita, riconosce solo le righe che fanno parte di una transazione confermata (riuscita), quindi inserisce il messaggio relativo all'evento nella coda SQS, che viene letta dal servizio di pagamento per un'ulteriore elaborazione. Questo comportamento risolve il problema delle operazioni di doppia scrittura e mantiene l'ordine dei messaggi e degli eventi utilizzando timestamp e numeri di sequenza.



Utilizzo dell'acquisizione dei dati di modifica (CDC)

Alcuni database supportano la pubblicazione di modifiche a livello di elemento per acquisire i dati modificati. È possibile identificare gli elementi modificati e inviare di conseguenza una notifica di evento. Ciò consente di risparmiare il sovraccarico dovuto alla creazione di un'altra tabella per tenere traccia degli aggiornamenti. L'evento avviato dal servizio in corso viene memorizzato in un altro attributo dello stesso elemento.

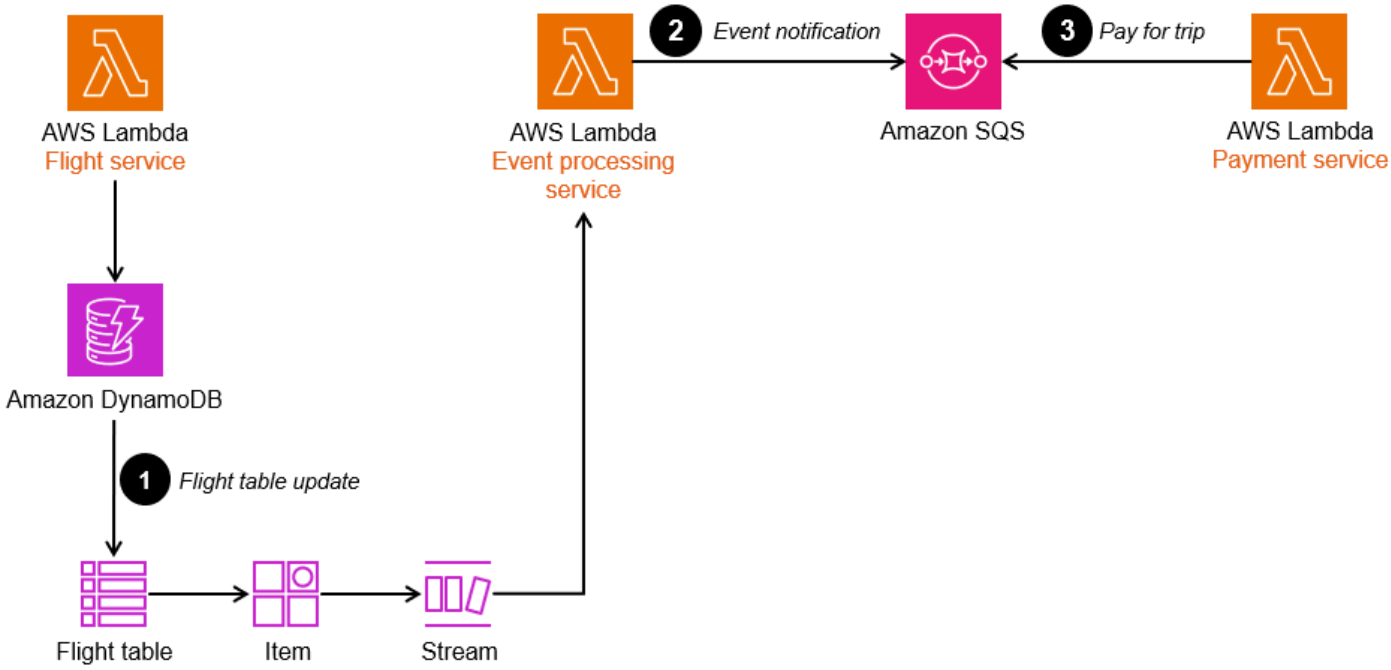
Amazon DynamoDB è un database NoSQL chiave-valore che supporta gli aggiornamenti CDC. Nel seguente diagramma di sequenza, DynamoDB pubblica modifiche a livello di elemento nei flussi Amazon DynamoDB. Il servizio di elaborazione degli eventi legge i flussi e pubblica la notifica dell'evento sul servizio di pagamento per un'ulteriore elaborazione.



I flussi DynamoDB acquisiscono il flusso di informazioni relative alle modifiche a livello di elemento in una tabella DynamoDB utilizzando una sequenza ordinata nel tempo.

È possibile implementare un modello di posta in uscita transazionale abilitando i flussi sulla tabella DynamoDB. La funzione Lambda per il servizio di elaborazione degli eventi è associata a questi flussi.

- Quando la tabella in corso viene aggiornata, i dati modificati vengono acquisiti dai flussi DynamoDB e il servizio di elaborazione degli eventi analizza il flusso alla ricerca di nuovi record.
- Quando diventano disponibili nuovi record di flusso, la funzione Lambda inserisce in modo sincrono il messaggio per l'evento nella coda SQS per un'ulteriore elaborazione. È possibile aggiungere un attributo all'elemento DynamoDB per acquisire il timestamp e il numero di sequenza necessari per migliorare la robustezza dell'implementazione.



Codice di esempio

Utilizzo di una tabella di posta in uscita

Il codice di esempio in questa sezione mostra come implementare il modello di posta in uscita transazionale utilizzando una tabella di posta in uscita. Per visualizzare il codice completo, consulta il GitHubrepository di questo esempio.

Il seguente frammento di codice salva l'entità `Flight` e l'evento `Flight` nel database nelle rispettive tabelle all'interno di una singola transazione.

```
@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 servizio separato si occupa di scansionare regolarmente la tabella in uscita alla ricerca di nuovi eventi, inviarli ad Amazon SQS ed eliminarli dalla tabella se Amazon SQS risponde correttamente. La frequenza di polling è configurabile nel file `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);
        }
    }
```

Utilizzo dell'acquisizione dei dati di modifica (CDC)

Il codice di esempio in questa sezione mostra come implementare il pattern transactional outbox utilizzando le funzionalità Change Data Capture (CDC) di DynamoDB. Per visualizzare il codice completo, consulta il repository di questo esempio. GitHub

Il seguente frammento di AWS Cloud Development Kit (AWS CDK) codice crea una tabella di volo DynamoDB e un flusso di dati Amazon Kinesis (`cdcStream`) e configura la tabella di volo per inviare tutti i relativi aggiornamenti allo stream.

```
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,
    }

});
```

Il frammento di codice e la configurazione seguenti definiscono una funzione Spring Cloud Stream che raccoglie gli aggiornamenti nel flusso Kinesis e inoltra questi eventi a una coda SQS per un'ulteriore elaborazione.

```
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 deposito

Per un'implementazione completa dell'architettura di esempio per questo pattern, consultate il GitHub repository all'indirizzo https://github.com/aws-samples/. transactional-outbox-pattern

Risorse

Riferimenti

- AWSCentro di architettura
- Centro sviluppatoriAWS
- La libreria Amazon Builders

Strumenti

- AWS Well-Architected Tool
- Contenitore AWS App2
- AWS Microservice Extractor for .NET

Metodologie

- L'app Twelve-Factor(ePub di Adam Wiggins)
- Michael T. NygardRilascialo! : Progettazione e implementazione di software pronto per la produzione. 2a ed. Raleigh, NC: Libreria pragmatica, 2018.
- Persistenza poliglotta(post sul blog di Martin Fowler)
- StranglerFigApplication(post sul blog di Martin Fowler)

Cronologia dei documenti

La tabella seguente descrive le modifiche significative apportate a questa guida. Per ricevere notifiche sugli aggiornamenti futuri, puoi abbonarti a un feed RSS.

Modifica	Descrizione	Data
Nuovi modelli	Sono stati aggiunti due nuovi pattern: architettura esagonale e scatter-gather.	7 maggio 2024
Nuovi esempi di codice	È stato aggiunto un codice di esempio per il caso d'uso Change Data Capture (CDC) al pattern transactional outbox.	23 febbraio 2024
Nuovi esempi di codice	• Il modello di posta in uscita transazionale è stato aggiornato con codice di esempio. • Rimossa la sezione sull'orch estrazione e i modelli coreografici, che sono stati sostituiti dalla coreografia saga e dall'orchestrazione saga.	16 novembre 2023
Nuovi modelli	Sono stati aggiunti tre nuovi modelli: coreografia saga, pubblicazione-sottoscrizione e approvvigionamento di eventi.	14 novembre 2023
Aggiorna	Aggiornata la sezione sull'implementazione del modello Strangler Fig.	2 ottobre 2023

Pubblicazione iniziale	Questa prima versione include otto modelli di progettazione: livello anticorruzione (ACL), routing API, interruttore, orchestrazione e coreografia, nuovo tentativo con backoff, orchestrazione saga, strangler fig e posta in uscita transazio nale.	28 luglio 2023

AWS Glossario delle linee guida prescrittive

I seguenti sono termini comunemente usati nelle strategie, nelle guide e nei modelli forniti da AWS Prescriptive Guidance. Per suggerire voci, utilizza il link Fornisci feedback alla fine del glossario.

Numeri

7 R

Sette strategie di migrazione comuni per trasferire le applicazioni sul cloud. Queste strategie si basano sulle 5 R identificate da Gartner nel 2011 e sono le seguenti:

- Rifattorizzare/riprogettare: trasferisci un'applicazione e modifica la sua architettura sfruttando appieno le funzionalità native del cloud per migliorare l'agilità, le prestazioni e la scalabilità. Ciò comporta in genere la portabilità del sistema operativo e del database. Esempio: esegui la migrazione del database Oracle on-premise ad Amazon Aurora edizione compatibile con PostgreSQL.
- Ridefinire la piattaforma (lift and reshape): trasferisci un'applicazione nel cloud e introduci un certo livello di ottimizzazione per sfruttare le funzionalità del cloud. Esempio: migra il tuo database Oracle locale su Amazon Relational Database Service (Amazon RDS) per Oracle nel cloud. AWS
- Riacquistare (drop and shop): passa a un prodotto diverso, in genere effettuando la transizione da una licenza tradizionale a un modello SaaS. Esempio: esegui la migrazione del tuo sistema di gestione delle relazioni con i clienti (CRM) su Salesforce.com.
- Eseguire il rehosting (lift and shift): trasferisci un'applicazione sul cloud senza apportare modifiche per sfruttare le funzionalità del cloud. Esempio: migra il tuo database Oracle locale su Oracle su un'istanza EC2 nel cloud. AWS
- Trasferire (eseguire il rehosting a livello hypervisor): trasferisci l'infrastruttura sul cloud senza acquistare nuovo hardware, riscrivere le applicazioni o modificare le operazioni esistenti. Questo scenario di migrazione è specifico di VMware Cloud on AWS, che supporta la compatibilità delle macchine virtuali (VM) e la portabilità del carico di lavoro tra l'ambiente locale e. AWSÈ possibile utilizzare le tecnologie VMware Cloud Foundation dai data center on-premise durante la migrazione dell'infrastruttura a VMware Cloud su AWS. Esempio: trasferisci l'hypervisor che ospita il database Oracle su VMware Cloud on. AWS
- Riesaminare (mantenere): mantieni le applicazioni nell'ambiente di origine. Queste potrebbero includere applicazioni che richiedono una rifattorizzazione significativa che desideri rimandare a

un momento successivo e applicazioni legacy che desideri mantenere, perché non vi è alcuna giustificazione aziendale per effettuarne la migrazione.

- Ritirare: disattiva o rimuovi le applicazioni che non sono più necessarie nell'ambiente di origine.

A

ABAC

Vedi controllo degli accessi basato sugli attributi.

servizi astratti

Vedi servizi gestiti.

ACIDO

Vedi atomicità, consistenza, isolamento, durata.

migrazione attiva-attiva

Un metodo di migrazione del database in cui i database di origine e di destinazione vengono mantenuti sincronizzati (utilizzando uno strumento di replica bidirezionale o operazioni di doppia scrittura) ed entrambi i database gestiscono le transazioni provenienti dalle applicazioni di connessione durante la migrazione. Questo metodo supporta la migrazione in piccoli batch controllati anziché richiedere una conversione una tantum. È più flessibile ma richiede più lavoro rispetto alla migrazione attiva-passiva.

migrazione attiva-passiva

Un metodo di migrazione di database in cui i database di origine e di destinazione vengono mantenuti sincronizzati, ma solo il database di origine gestisce le transazioni provenienti dalle applicazioni di connessione mentre i dati vengono replicati nel database di destinazione. Il database di destinazione non accetta alcuna transazione durante la migrazione.

funzione aggregata

Una funzione SQL che opera su un gruppo di righe e calcola un singolo valore restituito per il gruppo. Esempi di funzioni aggregate includono `SUM` e. `MAX`

Intelligenza artificiale

Vedi intelligenza artificiale.

AIOps

Guarda le operazioni di intelligenza artificiale.

anonimizzazione

Il processo di eliminazione permanente delle informazioni personali in un set di dati. L'anonimizzazione può aiutare a proteggere la privacy personale. I dati anonimi non sono più considerati dati personali.

anti-modello

Una soluzione utilizzata di frequente per un problema ricorrente in cui la soluzione è controproducente, inefficace o meno efficace di un'alternativa.

controllo delle applicazioni

Un approccio alla sicurezza che consente l'uso solo di applicazioni approvate per proteggere un sistema dal malware.

portfolio di applicazioni

Una raccolta di informazioni dettagliate su ogni applicazione utilizzata da un'organizzazione, compresi i costi di creazione e manutenzione dell'applicazione e il relativo valore aziendale. Queste informazioni sono fondamentali per il processo di scoperta e analisi del portfolio e aiutano a identificare e ad assegnare la priorità alle applicazioni da migrare, modernizzare e ottimizzare.

intelligenza artificiale (IA)

Il campo dell'informatica dedicato all'uso delle tecnologie informatiche per svolgere funzioni cognitive tipicamente associate agli esseri umani, come l'apprendimento, la risoluzione di problemi e il riconoscimento di schemi. Per ulteriori informazioni, consulta la sezione Che cos'è l'intelligenza artificiale?

operazioni di intelligenza artificiale (AIOps)

Il processo di utilizzo delle tecniche di machine learning per risolvere problemi operativi, ridurre gli incidenti operativi e l'intervento umano e aumentare la qualità del servizio. Per ulteriori informazioni su come viene utilizzato AIOps nella strategia di migrazione AWS , consulta la guida all'integrazione delle operazioni.

crittografia asimmetrica

Un algoritmo di crittografia che utilizza una coppia di chiavi, una chiave pubblica per la crittografia e una chiave privata per la decrittografia. Puoi condividere la chiave pubblica perché non viene utilizzata per la decrittografia, ma l'accesso alla chiave privata deve essere altamente limitato.

atomicità, consistenza, isolamento, durabilità (ACID)

Un insieme di proprietà del software che garantiscono la validità dei dati e l'affidabilità operativa di un database, anche in caso di errori, interruzioni di corrente o altri problemi.

Controllo degli accessi basato su attributi (ABAC)

La pratica di creare autorizzazioni dettagliate basate su attributi utente, come reparto, ruolo professionale e nome del team. Per ulteriori informazioni, consulta ABAC for AWS nella documentazione AWS Identity and Access Management (IAM).

fonte di dati autorevole

Una posizione in cui è archiviata la versione principale dei dati, considerata la fonte di informazioni più affidabile. È possibile copiare i dati dalla fonte di dati autorevole in altre posizioni allo scopo di elaborarli o modificarli, ad esempio anonimizzandoli, oscurandoli o pseudonimizzandoli.

Zona di disponibilità

Una posizione distinta all'interno di un edificio Regione AWS che è isolata dai guasti in altre zone di disponibilità e offre una connettività di rete economica e a bassa latenza verso altre zone di disponibilità nella stessa regione.

AWS Cloud Adoption Framework (CAF)AWS

Un framework di linee guida e best practice AWS per aiutare le organizzazioni a sviluppare un piano efficiente ed efficace per passare con successo al cloud. AWS CAF organizza le linee guida in sei aree di interesse chiamate prospettive: business, persone, governance, piattaforma, sicurezza e operazioni. Le prospettive relative ad azienda, persone e governance si concentrano sulle competenze e sui processi aziendali; le prospettive relative alla piattaforma, alla sicurezza e alle operazioni si concentrano sulle competenze e sui processi tecnici. Ad esempio, la prospettiva relativa alle persone si rivolge alle parti interessate che gestiscono le risorse umane (HR), le funzioni del personale e la gestione del personale. In questa prospettiva, AWS CAF fornisce linee guida per lo sviluppo delle persone, la formazione e le comunicazioni per aiutare a preparare l'organizzazione all'adozione del cloud di successo. Per ulteriori informazioni, consulta il sito web di AWS CAF e il white paper AWS CAF.

AWS Workload Qualification Framework (WQF)AWS

Uno strumento che valuta i carichi di lavoro di migrazione dei database, consiglia strategie di migrazione e fornisce stime del lavoro. AWS WQF è incluso in (). AWS Schema Conversion Tool AWS SCT Analizza gli schemi di database e gli oggetti di codice, il codice dell'applicazione, le dipendenze e le caratteristiche delle prestazioni e fornisce report di valutazione.

B

bot difettoso

Un bot che ha lo scopo di interrompere o causare danni a individui o organizzazioni.

BCP

Vedi la pianificazione della continuità operativa.

grafico comportamentale

Una vista unificata, interattiva dei comportamenti delle risorse e delle interazioni nel tempo. Puoi utilizzare un grafico comportamentale con Amazon Detective per esaminare tentativi di accesso non riusciti, chiamate API sospette e azioni simili. Per ulteriori informazioni, consulta Dati in un grafico comportamentale nella documentazione di Detective.

sistema big-endian

Un sistema che memorizza per primo il byte più importante. Vedi anche endianness.

Classificazione binaria

Un processo che prevede un risultato binario (una delle due classi possibili). Ad esempio, il modello di machine learning potrebbe dover prevedere problemi come "Questa e-mail è spam o non è spam?" o "Questo prodotto è un libro o un'auto?"

filtro Bloom

Una struttura di dati probabilistica ed efficiente in termini di memoria che viene utilizzata per verificare se un elemento fa parte di un set.

distribuzioni blu/verdi

Una strategia di implementazione in cui si creano due ambienti separati ma identici. La versione corrente dell'applicazione viene eseguita in un ambiente (blu) e la nuova versione

dell'applicazione nell'altro ambiente (verde). Questa strategia consente di ripristinare rapidamente il sistema con un impatto minimo.

bot

Un'applicazione software che esegue attività automatizzate su Internet e simula l'attività o l'interazione umana. Alcuni bot sono utili o utili, come i web crawler che indicizzano le informazioni su Internet. Alcuni altri bot, noti come bot dannosi, hanno lo scopo di disturbare o causare danni a individui o organizzazioni.

botnet

Reti di bot infettate da malware e controllate da un'unica parte, nota come bot herder o bot operator. Le botnet sono il meccanismo più noto per scalare i bot e il loro impatto.

ramo

Un'area contenuta di un repository di codice. Il primo ramo creato in un repository è il ramo principale. È possibile creare un nuovo ramo a partire da un ramo esistente e quindi sviluppare funzionalità o correggere bug al suo interno. Un ramo creato per sviluppare una funzionalità viene comunemente detto ramo di funzionalità. Quando la funzionalità è pronta per il rilascio, il ramo di funzionalità viene ricongiunto al ramo principale. Per ulteriori informazioni, consulta Informazioni sulle filiali (documentazione). GitHub

accesso break-glass

In circostanze eccezionali e tramite una procedura approvata, un mezzo rapido per consentire a un utente di accedere a un sito a Account AWS cui in genere non dispone delle autorizzazioni necessarie. Per ulteriori informazioni, vedere l'indicatore Implementate break-glass procedures nella guida Well-Architected AWS .

strategia brownfield

L'infrastruttura esistente nell'ambiente. Quando si adotta una strategia brownfield per un'architettura di sistema, si progetta l'architettura in base ai vincoli dei sistemi e dell'infrastruttura attuali. Per l'espansione dell'infrastruttura esistente, è possibile combinare strategie brownfield e greenfield.

cache del buffer

L'area di memoria in cui sono archiviati i dati a cui si accede con maggiore frequenza.

capacità di business

Azioni intraprese da un'azienda per generare valore (ad esempio vendite, assistenza clienti o marketing). Le architetture dei microservizi e le decisioni di sviluppo possono essere guidate dalle capacità aziendali. Per ulteriori informazioni, consulta la sezione Organizzazione in base alle funzionalità aziendali del whitepaper Esecuzione di microservizi containerizzati su AWS.

pianificazione della continuità operativa (BCP)

Un piano che affronta il potenziale impatto di un evento che comporta l'interruzione dell'attività, come una migrazione su larga scala, sulle operazioni e consente a un'azienda di riprendere rapidamente le operazioni.

C

CAF

Vedi AWS Cloud Adoption Framework.

implementazione canaria

Il rilascio lento e incrementale di una versione agli utenti finali. Quando sei sicuro, distribuisci la nuova versione e sostituisci la versione corrente nella sua interezza.

CoE

Vedi Cloud Center of Excellence.

CDC

Vedi Change Data Capture.

Change Data Capture (CDC)

Il processo di tracciamento delle modifiche a un'origine dati, ad esempio una tabella di database, e di registrazione dei metadati relativi alla modifica. È possibile utilizzare CDC per vari scopi, ad esempio il controllo o la replica delle modifiche in un sistema di destinazione per mantenere la sincronizzazione.

ingegneria del caos

Introduzione intenzionale di guasti o eventi dirompenti per testare la resilienza di un sistema. Puoi usare AWS Fault Injection Service (AWS FIS) per eseguire esperimenti che stressano i tuoi AWS carichi di lavoro e valutarne la risposta.

CI/CD

Vedi integrazione continua e distribuzione continua.

classificazione

Un processo di categorizzazione che aiuta a generare previsioni. I modelli di ML per problemi di classificazione prevedono un valore discreto. I valori discreti sono sempre distinti l'uno dall'altro. Ad esempio, un modello potrebbe dover valutare se in un'immagine è presente o meno un'auto.

crittografia lato client

Crittografia dei dati a livello locale, prima che il destinatario li Servizio AWS riceva.

centro di eccellenza del cloud (CCoE)

Un team multidisciplinare che guida le iniziative di adozione del cloud in tutta l'organizzazione, tra cui lo sviluppo di best practice per il cloud, la mobilitazione delle risorse, la definizione delle tempistiche di migrazione e la guida dell'organizzazione attraverso trasformazioni su larga scala. Per ulteriori informazioni, consulta i post di CCoE sul blog AWS Cloud Enterprise Strategy.

cloud computing

La tecnologia cloud generalmente utilizzata per l'archiviazione remota di dati e la gestione dei dispositivi IoT. Il cloud computing è generalmente connesso alla tecnologia di edge computing.

modello operativo cloud

In un'organizzazione IT, il modello operativo utilizzato per creare, maturare e ottimizzare uno o più ambienti cloud. Per ulteriori informazioni, consulta Building your Cloud Operating Model.

fasi di adozione del cloud

Le quattro fasi che le organizzazioni in genere attraversano quando migrano al AWS cloud:

- Progetto: esecuzione di alcuni progetti relativi al cloud per scopi di dimostrazione e apprendimento
- Fondamento: effettuare investimenti fondamentali per dimensionare l'adozione del cloud (ad esempio, creazione di una zona di destinazione, definizione di un CCoE, definizione di un modello operativo)
- Migrazione: migrazione di singole applicazioni
- Reinvenzione: ottimizzazione di prodotti e servizi e innovazione nel cloud

Queste fasi sono state definite da Stephen Orban nel post sul blog The Journey Toward Cloud-First & the Stages of Adoption on the AWS Cloud Enterprise Strategy. Per informazioni su come si relazionano alla strategia di AWS migrazione, consulta la guida alla preparazione alla migrazione.

CMDB

Vedi database di gestione della configurazione.

repository di codice

Una posizione in cui il codice di origine e altri asset, come documentazione, esempi e script, vengono archiviati e aggiornati attraverso processi di controllo delle versioni. Gli archivi cloud più comuni includono GitHub o AWS CodeCommit. Ogni versione del codice è denominata ramo. In una struttura a microservizi, ogni repository è dedicato a una singola funzionalità. Una singola pipeline CI/CD può utilizzare più repository.

cache fredda

Una cache del buffer vuota, non ben popolata o contenente dati obsoleti o irrilevanti. Ciò influisce sulle prestazioni perché l'istanza di database deve leggere dalla memoria o dal disco principale, il che richiede più tempo rispetto alla lettura dalla cache del buffer.

dati freddi

Dati a cui si accede raramente e che in genere sono storici. Quando si eseguono interrogazioni di questo tipo di dati, le interrogazioni lente sono in genere accettabili. Lo spostamento di questi dati su livelli o classi di storage meno costosi e con prestazioni inferiori può ridurre i costi.

visione artificiale (CV)

Un campo dell'intelligenza artificiale che utilizza l'apprendimento automatico per analizzare ed estrarre informazioni da formati visivi come immagini e video digitali. Ad esempio, AWS Panorama offre dispositivi che aggiungono CV alle reti di telecamere locali e Amazon SageMaker fornisce algoritmi di elaborazione delle immagini per CV.

deriva della configurazione

Per un carico di lavoro, una modifica della configurazione rispetto allo stato previsto. Potrebbe causare la non conformità del carico di lavoro e in genere è graduale e involontaria.

database di gestione della configurazione (CMDB)

Un repository che archivia e gestisce le informazioni su un database e il relativo ambiente IT, inclusi i componenti hardware e software e le relative configurazioni. In genere si utilizzano i dati di un CMDB nella fase di individuazione e analisi del portafoglio della migrazione.

Pacchetto di conformità

Una raccolta di AWS Config regole e azioni correttive che puoi assemblare per personalizzare i controlli di conformità e sicurezza. È possibile distribuire un pacchetto di conformità come singola entità in una regione Account AWS and o all'interno di un'organizzazione utilizzando un modello YAML. Per ulteriori informazioni, consulta i Conformance Pack nella documentazione. AWS Config

integrazione e distribuzione continua (continuous integration and continuous delivery, CI/CD)

Il processo di automazione delle fasi di origine, creazione, test, gestione temporanea e produzione del processo di rilascio del software. Il processo CI/CD è comunemente descritto come una pipeline. CI/CD può aiutare ad automatizzare i processi, migliorare la produttività, migliorare la qualità del codice e velocizzare le distribuzioni. Per ulteriori informazioni, consulta Vantaggi della distribuzione continua. CD può anche significare continuous deployment (implementazione continua). Per ulteriori informazioni, consulta Distribuzione continua e implementazione continua a confronto.

CV

Vedi visione artificiale.

D

dati a riposo

Dati stazionari nella rete, ad esempio i dati archiviati.

classificazione dei dati

Un processo per identificare e classificare i dati nella rete in base alla loro criticità e sensibilità. È un componente fondamentale di qualsiasi strategia di gestione dei rischi di sicurezza informatica perché consente di determinare i controlli di protezione e conservazione appropriati per i dati. La classificazione dei dati è un componente del pilastro della sicurezza nel AWS Well-Architected Framework. Per ulteriori informazioni, consulta Classificazione dei dati.

deriva dei dati

Una variazione significativa tra i dati di produzione e i dati utilizzati per addestrare un modello di machine learning o una modifica significativa dei dati di input nel tempo. La deriva dei dati può ridurre la qualità, l'accuratezza e l'equità complessive nelle previsioni dei modelli ML.

dati in transito

Dati che si spostano attivamente attraverso la rete, ad esempio tra le risorse di rete.

rete di dati

Un framework architettonico che fornisce la proprietà distribuita e decentralizzata dei dati con gestione e governance centralizzate.

riduzione al minimo dei dati

Il principio della raccolta e del trattamento dei soli dati strettamente necessari. Praticare la riduzione al minimo dei dati in the Cloud AWS può ridurre i rischi per la privacy, i costi e l'impronta di carbonio delle analisi.

perimetro dei dati

Una serie di barriere preventive nell' AWS ambiente che aiutano a garantire che solo le identità attendibili accedano alle risorse attendibili delle reti previste. Per ulteriori informazioni, consulta Building a data perimeter on. AWS

pre-elaborazione dei dati

Trasformare i dati grezzi in un formato che possa essere facilmente analizzato dal modello di ML. La pre-elaborazione dei dati può comportare la rimozione di determinate colonne o righe e l'eliminazione di valori mancanti, incoerenti o duplicati.

provenienza dei dati

Il processo di tracciamento dell'origine e della cronologia dei dati durante il loro ciclo di vita, ad esempio il modo in cui i dati sono stati generati, trasmessi e archiviati.

soggetto dei dati

Un individuo i cui dati vengono raccolti ed elaborati.

data warehouse

Un sistema di gestione dei dati che supporta la business intelligence, come l'analisi. I data warehouse contengono in genere grandi quantità di dati storici e vengono generalmente utilizzati per interrogazioni e analisi.

linguaggio di definizione del database (DDL)

Istruzioni o comandi per creare o modificare la struttura di tabelle e oggetti in un database.

linguaggio di manipolazione del database (DML)

Istruzioni o comandi per modificare (inserire, aggiornare ed eliminare) informazioni in un database.

DDL

Vedi linguaggio di definizione del database.

deep ensemble

Combinare più modelli di deep learning per la previsione. È possibile utilizzare i deep ensemble per ottenere una previsione più accurata o per stimare l'incertezza nelle previsioni.

deep learning

Un sottocampo del ML che utilizza più livelli di reti neurali artificiali per identificare la mappatura tra i dati di input e le variabili target di interesse.

defense-in-depth

Un approccio alla sicurezza delle informazioni in cui una serie di meccanismi e controlli di sicurezza sono accuratamente stratificati su una rete di computer per proteggere la riservatezza, l'integrità e la disponibilità della rete e dei dati al suo interno. Quando si adotta questa strategia AWS, si aggiungono più controlli a diversi livelli della AWS Organizations struttura per proteggere le risorse. Ad esempio, un defense-in-depth approccio potrebbe combinare l'autenticazione a più fattori, la segmentazione della rete e la crittografia.

amministratore delegato

In AWS Organizations, un servizio compatibile può registrare un account AWS membro per amministrare gli account dell'organizzazione e gestire le autorizzazioni per quel servizio. Questo account è denominato amministratore delegato per quel servizio specifico. Per ulteriori informazioni e un elenco di servizi compatibili, consulta Servizi che funzionano con AWS Organizations nella documentazione di AWS Organizations .

implementazione

Il processo di creazione di un'applicazione, di nuove funzionalità o di correzioni di codice disponibili nell'ambiente di destinazione. L'implementazione prevede l'applicazione di modifiche in una base di codice, seguita dalla creazione e dall'esecuzione di tale base di codice negli ambienti applicativi.

Ambiente di sviluppo

Vedi ambiente.

controllo di rilevamento

Un controllo di sicurezza progettato per rilevare, registrare e avvisare dopo che si è verificato un evento. Questi controlli rappresentano una seconda linea di difesa e avvisano l'utente in caso di eventi di sicurezza che aggirano i controlli preventivi in vigore. Per ulteriori informazioni, consulta Controlli di rilevamento in Implementazione dei controlli di sicurezza in AWS.

mappatura del flusso di valore dello sviluppo (DVSM)

Un processo utilizzato per identificare e dare priorità ai vincoli che influiscono negativamente sulla velocità e sulla qualità nel ciclo di vita dello sviluppo del software. DVSM estende il processo di mappatura del flusso di valore originariamente progettato per pratiche di produzione snella. Si concentra sulle fasi e sui team necessari per creare e trasferire valore attraverso il processo di sviluppo del software.

gemello digitale

Una rappresentazione virtuale di un sistema reale, ad esempio un edificio, una fabbrica, un'attrezzatura industriale o una linea di produzione. I gemelli digitali supportano la manutenzione predittiva, il monitoraggio remoto e l'ottimizzazione della produzione.

tabella delle dimensioni

In uno schema a stella, una tabella più piccola che contiene gli attributi dei dati quantitativi in una tabella dei fatti. Gli attributi della tabella delle dimensioni sono in genere campi di testo o numeri discreti che si comportano come testo. Questi attributi vengono comunemente utilizzati per il vincolo delle query, il filtraggio e l'etichettatura dei set di risultati.

disastro

Un evento che impedisce a un carico di lavoro o a un sistema di raggiungere gli obiettivi aziendali nella sua sede principale di implementazione. Questi eventi possono essere disastri naturali, guasti tecnici o il risultato di azioni umane, come errori di configurazione involontari o attacchi di malware.

disaster recovery (DR)

La strategia e il processo utilizzati per ridurre al minimo i tempi di inattività e la perdita di dati causati da un disastro. Per ulteriori informazioni, consulta Disaster Recovery of Workloads su AWS: Recovery in the Cloud in the AWS Well-Architected Framework.

DML

Vedi linguaggio di manipolazione del database.

progettazione basata sul dominio

Un approccio allo sviluppo di un sistema software complesso collegandone i componenti a domini in evoluzione, o obiettivi aziendali principali, perseguiti da ciascun componente. Questo concetto è stato introdotto da Eric Evans nel suo libro,Domain-Driven Design: Tackling Complexity in the Heart of Software (Boston: Addison-Wesley Professional, 2003). Per informazioni su come utilizzare la progettazione basata sul dominio con il modello del fico strangolatore (Strangler Fig), consulta la sezione Modernizzazione incrementale dei servizi Web Microsoft ASP.NET (ASMX) legacy utilizzando container e il Gateway Amazon API.

DOTT.

Vedi disaster recovery.

rilevamento della deriva

Tracciamento delle deviazioni da una configurazione di base. Ad esempio, puoi utilizzarlo AWS CloudFormation per rilevare la deriva nelle risorse di sistema oppure puoi usarlo AWS Control Tower per rilevare cambiamenti nella tua landing zone che potrebbero influire sulla conformità ai requisiti di governance.

DVSM

Vedi la mappatura del flusso di valore dello sviluppo.

E

EDA

Vedi analisi esplorativa dei dati.

edge computing

La tecnologia che aumenta la potenza di calcolo per i dispositivi intelligenti all'edge di una rete IoT. Rispetto al cloud computing, l'edge computing può ridurre la latenza di comunicazione e migliorare i tempi di risposta.

crittografia

Un processo di elaborazione che trasforma i dati in chiaro, leggibili dall'uomo, in testo cifrato.

chiave crittografica

Una stringa crittografica di bit randomizzati generata da un algoritmo di crittografia. Le chiavi possono variare di lunghezza e ogni chiave è progettata per essere imprevedibile e univoca.

endianità

L'ordine in cui i byte vengono archiviati nella memoria del computer. I sistemi big-endian memorizzano per primo il byte più importante. I sistemi little-endian memorizzano per primo il byte meno importante.

endpoint

Vedi service endpoint.

servizio endpoint

Un servizio che puoi ospitare in un cloud privato virtuale (VPC) da condividere con altri utenti. Puoi creare un servizio endpoint con AWS PrivateLink e concedere autorizzazioni ad altri Account AWS o a AWS Identity and Access Management (IAM) principali. Questi account o principali possono connettersi al servizio endpoint in privato creando endpoint VPC di interfaccia. Per ulteriori informazioni, consulta Creazione di un servizio endpoint nella documentazione di Amazon Virtual Private Cloud (Amazon VPC).

pianificazione delle risorse aziendali (ERP)

Un sistema che automatizza e gestisce i processi aziendali chiave (come contabilità, MES e gestione dei progetti) per un'azienda.

crittografia envelope

Il processo di crittografia di una chiave di crittografia con un'altra chiave di crittografia. Per ulteriori informazioni, vedete Envelope encryption nella documentazione AWS Key Management Service (AWS KMS).

ambiente

Un'istanza di un'applicazione in esecuzione. Di seguito sono riportati i tipi di ambiente più comuni nel cloud computing:

- ambiente di sviluppo: un'istanza di un'applicazione in esecuzione disponibile solo per il team principale responsabile della manutenzione dell'applicazione. Gli ambienti di sviluppo vengono utilizzati per testare le modifiche prima di promuoverle negli ambienti superiori. Questo tipo di ambiente viene talvolta definito ambiente di test.

- ambienti inferiori: tutti gli ambienti di sviluppo di un'applicazione, ad esempio quelli utilizzati per le build e i test iniziali.
- ambiente di produzione: un'istanza di un'applicazione in esecuzione a cui gli utenti finali possono accedere. In una pipeline CI/CD, l'ambiente di produzione è l'ultimo ambiente di implementazione.
- ambienti superiori: tutti gli ambienti a cui possono accedere utenti diversi dal team di sviluppo principale. Si può trattare di un ambiente di produzione, ambienti di preproduzione e ambienti per i test di accettazione da parte degli utenti.

epica

Nelle metodologie agili, categorie funzionali che aiutano a organizzare e dare priorità al lavoro. Le epiche forniscono una descrizione di alto livello dei requisiti e delle attività di implementazione. Ad esempio, le epopee della sicurezza AWS CAF includono la gestione delle identità e degli accessi, i controlli investigativi, la sicurezza dell'infrastruttura, la protezione dei dati e la risposta agli incidenti. Per ulteriori informazioni sulle epiche, consulta la strategia di migrazione AWS , consulta la guida all'implementazione del programma.

ERP

Vedi pianificazione delle risorse aziendali.

analisi esplorativa dei dati (EDA)

Il processo di analisi di un set di dati per comprenderne le caratteristiche principali. Si raccolgono o si aggregano dati e quindi si eseguono indagini iniziali per trovare modelli, rilevare anomalie e verificare ipotesi. L'EDA viene eseguita calcolando statistiche di riepilogo e creando visualizzazioni di dati.

F

tabella dei fatti

Il tavolo centrale in uno schema a stella. Memorizza dati quantitativi sulle operazioni aziendali. In genere, una tabella dei fatti contiene due tipi di colonne: quelle che contengono misure e quelle che contengono una chiave esterna per una tabella di dimensioni.

fallire velocemente

Una filosofia che utilizza test frequenti e incrementali per ridurre il ciclo di vita dello sviluppo. È una parte fondamentale di un approccio agile.

limite di isolamento dei guasti

Nel Cloud AWS, un limite come una zona di disponibilità Regione AWS, un piano di controllo o un piano dati che limita l'effetto di un errore e aiuta a migliorare la resilienza dei carichi di lavoro. Per ulteriori informazioni, consulta AWS Fault Isolation Boundaries.

ramo di funzionalità

Vedi filiale.

caratteristiche

I dati di input che usi per fare una previsione. Ad esempio, in un contesto di produzione, le caratteristiche potrebbero essere immagini acquisite periodicamente dalla linea di produzione.

importanza delle caratteristiche

Quanto è importante una caratteristica per le previsioni di un modello. Di solito viene espresso come punteggio numerico che può essere calcolato con varie tecniche, come Shapley Additive Explanations (SHAP) e gradienti integrati. Per ulteriori informazioni, vedere Interpretabilità del modello di machine learning con:AWS.

trasformazione delle funzionalità

Per ottimizzare i dati per il processo di machine learning, incluso l'arricchimento dei dati con fonti aggiuntive, il dimensionamento dei valori o l'estrazione di più set di informazioni da un singolo campo di dati. Ciò consente al modello di ML di trarre vantaggio dai dati. Ad esempio, se suddividi la data "2021-05-27 00:15:37" in "2021", "maggio", "giovedì" e "15", puoi aiutare l'algoritmo di apprendimento ad apprendere modelli sfumati associati a diversi componenti dei dati.

FGAC

Vedi il controllo granulare degli accessi.

controllo granulare degli accessi (FGAC)

L'uso di più condizioni per consentire o rifiutare una richiesta di accesso.

migrazione flash-cut

Un metodo di migrazione del database che utilizza la replica continua dei dati tramite l'acquisizione dei dati delle modifiche per migrare i dati nel più breve tempo possibile, anziché utilizzare un approccio graduale. L'obiettivo è ridurre al minimo i tempi di inattività.

G

blocco geografico

Vedi restrizioni geografiche.

limitazioni geografiche (blocco geografico)

In Amazon CloudFront, un'opzione per impedire agli utenti di determinati paesi di accedere alle distribuzioni di contenuti. Puoi utilizzare un elenco consentito o un elenco di blocco per specificare i paesi approvati e vietati. Per ulteriori informazioni, consulta Limitare la distribuzione geografica dei contenuti nella CloudFront documentazione.

Flusso di lavoro di GitFlow

Un approccio in cui gli ambienti inferiori e superiori utilizzano rami diversi in un repository di codice di origine. Il flusso di lavoro Gitflow è considerato obsoleto e il flusso di lavoro basato su trunk è l'approccio moderno e preferito.

strategia greenfield

L'assenza di infrastrutture esistenti in un nuovo ambiente. Quando si adotta una strategia greenfield per un'architettura di sistema, è possibile selezionare tutte le nuove tecnologie senza il vincolo della compatibilità con l'infrastruttura esistente, nota anche come brownfield. Per l'espansione dell'infrastruttura esistente, è possibile combinare strategie brownfield e greenfield.

guardrail

Una regola di livello elevato che consente di governare risorse, policy e conformità tra le unità organizzative (OU). I guardrail preventivi applicano le policy per garantire l'allineamento agli standard di conformità. Vengono implementati utilizzando le policy di controllo dei servizi e i limiti delle autorizzazioni IAM. I guardrail di rilevamento rilevano le violazioni delle policy e i problemi di conformità e generano avvisi per porvi rimedio. Sono implementati utilizzando Amazon AWS Config AWS Security Hub GuardDuty AWS Trusted Advisor, Amazon Inspector e controlli personalizzati AWS Lambda .

H

AH

Vedi disponibilità elevata.

migrazione di database eterogenea

Migrazione del database di origine in un database di destinazione che utilizza un motore di database diverso (ad esempio, da Oracle ad Amazon Aurora). La migrazione eterogenea fa in genere parte di uno sforzo di riprogettazione e la conversione dello schema può essere un'attività complessa. AWS offre AWS SCT che aiuta con le conversioni dello schema.

alta disponibilità (HA)

La capacità di un carico di lavoro di funzionare in modo continuo, senza intervento, in caso di sfide o disastri. I sistemi HA sono progettati per il failover automatico, fornire costantemente prestazioni di alta qualità e gestire carichi e guasti diversi con un impatto minimo sulle prestazioni.

modernizzazione storica

Un approccio utilizzato per modernizzare e aggiornare i sistemi di tecnologia operativa (OT) per soddisfare meglio le esigenze dell'industria manifatturiera. Uno storico è un tipo di database utilizzato per raccogliere e archiviare dati da varie fonti in una fabbrica.

migrazione di database omogenea

Migrazione del database di origine in un database di destinazione che condivide lo stesso motore di database (ad esempio, da Microsoft SQL Server ad Amazon RDS per SQL Server). La migrazione omogenea fa in genere parte di un'operazione di rehosting o ridefinizione della piattaforma. Per migrare lo schema è possibile utilizzare le utilità native del database.

dati caldi

Dati a cui si accede frequentemente, come dati in tempo reale o dati di traduzione recenti. Questi dati richiedono in genere un livello o una classe di storage ad alte prestazioni per fornire risposte rapide alle query.

hotfix

Una soluzione urgente per un problema critico in un ambiente di produzione. A causa della sua urgenza, un hotfix viene in genere creato al di fuori del tipico DevOps flusso di lavoro di rilascio.

periodo di hypercare

Subito dopo la conversione, il periodo di tempo in cui un team di migrazione gestisce e monitora le applicazioni migrate nel cloud per risolvere eventuali problemi. In genere, questo periodo dura da 1 a 4 giorni. Al termine del periodo di hypercare, il team addetto alla migrazione in genere trasferisce la responsabilità delle applicazioni al team addetto alle operazioni cloud.

I

IaC

Considera l'infrastruttura come codice.

Policy basata su identità

Una policy associata a uno o più principi IAM che definisce le relative autorizzazioni all'interno dell' Cloud AWS ambiente.

applicazione inattiva

Un'applicazione che prevede un uso di CPU e memoria medio compreso tra il 5% e il 20% in un periodo di 90 giorni. In un progetto di migrazione, è normale ritirare queste applicazioni o mantenerle on-premise.

IIoT

Vedi Industrial Internet of Things.

infrastruttura immutabile

Un modello che implementa una nuova infrastruttura per i carichi di lavoro di produzione anziché aggiornare, applicare patch o modificare l'infrastruttura esistente. Le infrastrutture immutabili sono intrinsecamente più coerenti, affidabili e prevedibili delle infrastrutture mutabili. Per ulteriori informazioni, consulta la best practice Deploy using immutable infrastructure in Well-Architected AWS Framework.

VPC in ingresso (ingress)

In un'architettura AWS multi-account, un VPC che accetta, ispeziona e indirizza le connessioni di rete dall'esterno di un'applicazione. Nel documento Architettura di riferimento per la sicurezza di AWS si consiglia di configurare l'account di rete con VPC in entrata, in uscita e di ispezione per proteggere l'interfaccia bidirezionale tra l'applicazione e Internet in generale.

migrazione incrementale

Una strategia di conversione in cui si esegue la migrazione dell'applicazione in piccole parti anziché eseguire una conversione singola e completa. Ad esempio, inizialmente potresti spostare solo alcuni microservizi o utenti nel nuovo sistema. Dopo aver verificato che tutto funzioni correttamente, puoi spostare in modo incrementale microservizi o utenti aggiuntivi fino alla disattivazione del sistema legacy. Questa strategia riduce i rischi associati alle migrazioni di grandi dimensioni.

Industria 4.0

Un termine introdotto da Klaus Schwab nel 2016 per riferirsi alla modernizzazione dei processi di produzione attraverso progressi in termini di connettività, dati in tempo reale, automazione, analisi e AI/ML.

infrastruttura

Tutte le risorse e gli asset contenuti nell'ambiente di un'applicazione.

infrastruttura come codice (IaC)

Il processo di provisioning e gestione dell'infrastruttura di un'applicazione tramite un insieme di file di configurazione. Il processo IaC è progettato per aiutarti a centralizzare la gestione dell'infrastruttura, a standardizzare le risorse e a dimensionare rapidamente, in modo che i nuovi ambienti siano ripetibili, affidabili e coerenti.

Internet delle cose industriale (IIoT)

L'uso di sensori e dispositivi connessi a Internet nei settori industriali, come quello manifatturiero, energetico, automobilistico, sanitario, delle scienze della vita e dell'agricoltura. Per ulteriori informazioni, consulta Creazione di una strategia di trasformazione digitale dell'Internet delle cose industriale (IIoT).

VPC di ispezione

In un'architettura AWS multi-account, un VPC centralizzato che gestisce le ispezioni del traffico di rete tra VPC (uguali o diversi Regioni AWS), Internet e reti locali. Nel documento Architettura di riferimento per la sicurezza di AWS si consiglia di configurare l'account di rete con VPC in entrata, in uscita e di ispezione per proteggere l'interfaccia bidirezionale tra l'applicazione e Internet in generale.

Internet of Things (IoT)

La rete di oggetti fisici connessi con sensori o processori incorporati che comunicano con altri dispositivi e sistemi tramite Internet o una rete di comunicazione locale. Per ulteriori informazioni, consulta Cos'è l'IoT?

interpretabilità

Una caratteristica di un modello di machine learning che descrive il grado in cui un essere umano è in grado di comprendere in che modo le previsioni del modello dipendono dai suoi input. Per ulteriori informazioni, consulta la sezione Interpretabilità dei modelli di machine learning con AWS.

IoT

Vedi Internet of Things.

libreria di informazioni IT (ITIL)

Una serie di best practice per offrire servizi IT e allinearli ai requisiti aziendali. ITIL fornisce le basi per ITSM.

gestione dei servizi IT (ITSM)

Attività associate alla progettazione, implementazione, gestione e supporto dei servizi IT per un'organizzazione. Per informazioni sull'integrazione delle operazioni cloud con gli strumenti ITSM, consulta la guida all'integrazione delle operazioni.

ITIL

Vedi la libreria di informazioni IT.

ITSM

Vedi Gestione dei servizi IT.

L

controllo degli accessi basato su etichette (LBAC)

Un'implementazione del controllo di accesso obbligatorio (MAC) in cui agli utenti e ai dati stessi viene assegnato esplicitamente un valore di etichetta di sicurezza. L'intersezione tra l'etichetta di sicurezza utente e l'etichetta di sicurezza dei dati determina quali righe e colonne possono essere visualizzate dall'utente.

zona di destinazione

Una landing zone è un AWS ambiente multi-account ben progettato, scalabile e sicuro. Questo è un punto di partenza dal quale le organizzazioni possono avviare e distribuire rapidamente carichi di lavoro e applicazioni con fiducia nel loro ambiente di sicurezza e infrastruttura. Per ulteriori informazioni sulle zone di destinazione, consulta la sezione Configurazione di un ambiente AWS multi-account sicuro e scalabile.

migrazione su larga scala

Una migrazione di 300 o più server.

BIANCO

Vedi controllo degli accessi basato su etichette.

Privilegio minimo

La best practice di sicurezza per la concessione delle autorizzazioni minime richieste per eseguire un'attività. Per ulteriori informazioni, consulta Applicazione delle autorizzazioni del privilegio minimo nella documentazione di IAM.

eseguire il rehosting (lift and shift)

Vedi 7 R.

sistema little-endian

Un sistema che memorizza per primo il byte meno importante. Vedi anche endianità.

ambienti inferiori

Vedi ambiente.

M

machine learning (ML)

Un tipo di intelligenza artificiale che utilizza algoritmi e tecniche per il riconoscimento e l'apprendimento di schemi. Il machine learning analizza e apprende dai dati registrati, come i dati dell'Internet delle cose (IoT), per generare un modello statistico basato su modelli. Per ulteriori informazioni, consulta la sezione Machine learning.

ramo principale

Vedi filiale.

malware

Software progettato per compromettere la sicurezza o la privacy del computer. Il malware potrebbe interrompere i sistemi informatici, divulgare informazioni sensibili o ottenere accessi non autorizzati. Esempi di malware includono virus, worm, ransomware, trojan horse, spyware e keylogger.

servizi gestiti

Servizi AWS per cui AWS gestisce il livello di infrastruttura, il sistema operativo e le piattaforme e si accede agli endpoint per archiviare e recuperare i dati. Amazon Simple Storage Service (Amazon S3) Simple Storage Service (Amazon S3) e Amazon DynamoDB sono esempi di servizi gestiti. Questi sono noti anche come servizi astratti.

sistema di esecuzione della produzione (MES)

Un sistema software per tracciare, monitorare, documentare e controllare i processi di produzione che convertono le materie prime in prodotti finiti in officina.

MAP

Vedi Migration Acceleration Program.

meccanismo

Un processo completo in cui si crea uno strumento, si promuove l'adozione dello strumento e quindi si esaminano i risultati per apportare le modifiche. Un meccanismo è un ciclo che si rafforza e si migliora man mano che funziona. Per ulteriori informazioni, consulta Creazione di meccanismi nel AWS Well-Architected Framework.

account membro

Tutti gli account Account AWS diversi dall'account di gestione che fanno parte di un'organizzazione in. AWS Organizations Un account può essere membro di una sola organizzazione alla volta.

MEH

Vedi sistema di esecuzione della produzione.

Message Queuing Telemetry Transport (MQTT)

Un protocollo di comunicazione machine-to-machine (M2M) leggero, basato sul modello di pubblicazione/sottoscrizione, per dispositivi IoT con risorse limitate.

microservizio

Un piccolo servizio indipendente che comunica tramite API ben definite ed è in genere di proprietà di piccoli team autonomi. Ad esempio, un sistema assicurativo potrebbe includere microservizi che si riferiscono a funzionalità aziendali, come vendite o marketing, o sottodomini, come acquisti, reclami o analisi. I vantaggi dei microservizi includono agilità, dimensionamento flessibile,

facilità di implementazione, codice riutilizzabile e resilienza. Per ulteriori informazioni, consulta Integrazione dei microservizi utilizzando servizi serverless. AWS

architettura di microservizi

Un approccio alla creazione di un'applicazione con componenti indipendenti che eseguono ogni processo applicativo come microservizio. Questi microservizi comunicano tramite un'interfaccia ben definita utilizzando API leggere. Ogni microservizio in questa architettura può essere aggiornato, distribuito e dimensionato per soddisfare la richiesta di funzioni specifiche di un'applicazione. Per ulteriori informazioni, vedere Implementazione dei microservizi su. AWS

Programma di accelerazione della migrazione (MAP)

Un AWS programma che fornisce consulenza, supporto, formazione e servizi per aiutare le organizzazioni a costruire una solida base operativa per il passaggio al cloud e per contribuire a compensare il costo iniziale delle migrazioni. MAP include una metodologia di migrazione per eseguire le migrazioni precedenti in modo metodico e un set di strumenti per automatizzare e accelerare gli scenari di migrazione comuni.

migrazione su larga scala

Il processo di trasferimento della maggior parte del portfolio di applicazioni sul cloud avviene a ondate, con più applicazioni trasferite a una velocità maggiore in ogni ondata. Questa fase utilizza le migliori pratiche e le lezioni apprese nelle fasi precedenti per implementare una fabbrica di migrazione di team, strumenti e processi per semplificare la migrazione dei carichi di lavoro attraverso l'automazione e la distribuzione agile. Questa è la terza fase della strategia di migrazione AWS.

fabbrica di migrazione

Team interfunzionali che semplificano la migrazione dei carichi di lavoro attraverso approcci automatizzati e agili. I team di Migration Factory includono in genere operazioni, analisti e proprietari aziendali, ingegneri addetti alla migrazione, sviluppatori e DevOps professionisti che lavorano nell'ambito degli sprint. Tra il 20% e il 50% di un portfolio di applicazioni aziendali è costituito da schemi ripetuti che possono essere ottimizzati con un approccio di fabbrica. Per ulteriori informazioni, consulta la discussione sulle fabbriche di migrazione e la Guida alla fabbrica di migrazione al cloud in questo set di contenuti.

metadati di migrazione

Le informazioni sull'applicazione e sul server necessarie per completare la migrazione. Ogni modello di migrazione richiede un set diverso di metadati di migrazione. Esempi di metadati di migrazione includono la sottorete, il gruppo di sicurezza e l'account di destinazione. AWS

modello di migrazione

Un'attività di migrazione ripetibile che descrive in dettaglio la strategia di migrazione, la destinazione della migrazione e l'applicazione o il servizio di migrazione utilizzati. Esempio: riorganizza la migrazione su Amazon EC2 AWS con Application Migration Service.

Valutazione del portfolio di migrazione (MPA)

Uno strumento online che fornisce informazioni per la convalida del business case per la migrazione al Cloud. AWS MPA offre una valutazione dettagliata del portfolio (dimensionamento corretto dei server, prezzi, confronto del TCO, analisi dei costi di migrazione) e pianificazione della migrazione (analisi e raccolta dei dati delle applicazioni, raggruppamento delle applicazioni, prioritizzazione delle migrazioni e pianificazione delle ondate). Lo strumento MPA (richiede l'accesso) è disponibile gratuitamente per tutti i AWS consulenti e i consulenti dei partner APN.

valutazione della preparazione alla migrazione (MRA)

Il processo di acquisizione di informazioni sullo stato di preparazione al cloud di un'organizzazione, l'identificazione dei punti di forza e di debolezza e la creazione di un piano d'azione per colmare le lacune identificate, utilizzando il CAF. AWS Per ulteriori informazioni, consulta la guida di preparazione alla migrazione. MRA è la prima fase della strategia di migrazione AWS.

strategia di migrazione

L'approccio utilizzato per migrare un carico di lavoro nel cloud. AWS Per ulteriori informazioni, consulta la voce 7 R in questo glossario e consulta Mobilita la tua organizzazione per accelerare le migrazioni su larga scala.

ML

Vedi machine learning.

modernizzazione

Trasformazione di un'applicazione obsoleta (legacy o monolitica) e della relativa infrastruttura in un sistema agile, elastico e altamente disponibile nel cloud per ridurre i costi, aumentare

l'efficienza e sfruttare le innovazioni. Per ulteriori informazioni, vedere Strategia per la modernizzazione delle applicazioni in. Cloud AWS

valutazione della preparazione alla modernizzazione

Una valutazione che aiuta a determinare la preparazione alla modernizzazione delle applicazioni di un'organizzazione, identifica vantaggi, rischi e dipendenze e determina in che misura l'organizzazione può supportare lo stato futuro di tali applicazioni. Il risultato della valutazione è uno schema dell'architettura di destinazione, una tabella di marcia che descrive in dettaglio le fasi di sviluppo e le tappe fondamentali del processo di modernizzazione e un piano d'azione per colmare le lacune identificate. Per ulteriori informazioni, consulta la sezione Valutazione della preparazione alla modernizzazione per le applicazioni nel cloud AWS.

applicazioni monolitiche (monoliti)

Applicazioni eseguite come un unico servizio con processi strettamente collegati. Le applicazioni monolitiche presentano diversi inconvenienti. Se una funzionalità dell'applicazione registra un picco di domanda, l'intera architettura deve essere dimensionata. L'aggiunta o il miglioramento delle funzionalità di un'applicazione monolitica diventa inoltre più complessa man mano che la base di codice cresce. Per risolvere questi problemi, puoi utilizzare un'architettura di microservizi. Per ulteriori informazioni, consulta la sezione Scomposizione dei monoliti in microservizi.

MAPPA

Vedi Migration Portfolio Assessment.

MQTT

Vedi Message Queuing Telemetry Transport.

classificazione multiclasse

Un processo che aiuta a generare previsioni per più classi (prevedendo uno o più di due risultati). Ad esempio, un modello di machine learning potrebbe chiedere "Questo prodotto è un libro, un'auto o un telefono?" oppure "Quale categoria di prodotti è più interessante per questo cliente?"

infrastruttura mutabile

Un modello che aggiorna e modifica l'infrastruttura esistente per i carichi di lavoro di produzione. Per migliorare la coerenza, l'affidabilità e la prevedibilità, il AWS Well-Architected Framework consiglia l'uso di un'infrastruttura immutabile come best practice.

O

OAC

Vedi Origin Access Control.

QUERCIA

Vedi Origin Access Identity.

OCM

Vedi gestione delle modifiche organizzative.

migrazione offline

Un metodo di migrazione in cui il carico di lavoro di origine viene eliminato durante il processo di migrazione. Questo metodo prevede tempi di inattività prolungati e viene in genere utilizzato per carichi di lavoro piccoli e non critici.

OI

Vedi l'integrazione delle operazioni.

OLA

Vedi accordo a livello operativo.

migrazione online

Un metodo di migrazione in cui il carico di lavoro di origine viene copiato sul sistema di destinazione senza essere messo offline. Le applicazioni connesse al carico di lavoro possono continuare a funzionare durante la migrazione. Questo metodo comporta tempi di inattività pari a zero o comunque minimi e viene in genere utilizzato per carichi di lavoro di produzione critici.

OPC-UA

Vedi Open Process Communications - Unified Architecture.

Comunicazioni a processo aperto - Architettura unificata (OPC-UA)

Un protocollo di comunicazione machine-to-machine (M2M) per l'automazione industriale. OPC-UA fornisce uno standard di interoperabilità con schemi di crittografia, autenticazione e autorizzazione dei dati.

accordo a livello operativo (OLA)

Un accordo che chiarisce quali sono gli impegni reciproci tra i gruppi IT funzionali, a supporto di un accordo sul livello di servizio (SLA).

revisione della prontezza operativa (ORR)

Un elenco di domande e best practice associate che aiutano a comprendere, valutare, prevenire o ridurre la portata degli incidenti e dei possibili guasti. Per ulteriori informazioni, vedere Operational Readiness Reviews (ORR) nel Well-Architected AWS Framework.

tecnologia operativa (OT)

Sistemi hardware e software che interagiscono con l'ambiente fisico per controllare le operazioni, le apparecchiature e le infrastrutture industriali. Nella produzione, l'integrazione di sistemi OT e di tecnologia dell'informazione (IT) è un obiettivo chiave per le trasformazioni dell'Industria 4.0.

integrazione delle operazioni (OI)

Il processo di modernizzazione delle operazioni nel cloud, che prevede la pianificazione, l'automazione e l'integrazione della disponibilità. Per ulteriori informazioni, consulta la guida all'integrazione delle operazioni.

trail organizzativo

Un percorso creato da noi AWS CloudTrail che registra tutti gli eventi di un'organizzazione per tutti Account AWS . AWS Organizations Questo percorso viene creato in ogni Account AWS che fa parte dell'organizzazione e tiene traccia dell'attività in ogni account. Per ulteriori informazioni, consulta Creazione di un percorso per un'organizzazione nella CloudTrail documentazione.

gestione del cambiamento organizzativo (OCM)

Un framework per la gestione di trasformazioni aziendali importanti e che comportano l'interruzione delle attività dal punto di vista delle persone, della cultura e della leadership. OCM aiuta le organizzazioni a prepararsi e passare a nuovi sistemi e strategie accelerando l'adozione del cambiamento, affrontando i problemi di transizione e promuovendo cambiamenti culturali e organizzativi. Nella strategia di AWS migrazione, questo framework si chiama accelerazione delle persone, a causa della velocità di cambiamento richiesta nei progetti di adozione del cloud. Per ulteriori informazioni, consultare la Guida OCM.

controllo dell'accesso all'origine (OAC)

In CloudFront, un'opzione avanzata per limitare l'accesso per proteggere i contenuti di Amazon Simple Storage Service (Amazon S3). OAC supporta tutti i bucket S3 in generale Regioni AWS, la

crittografia lato server con AWS KMS (SSE-KMS) e le richieste dinamiche e dirette al bucket S3. `PUT` `DELETE`

identità di accesso origine (OAI)

Nel CloudFront, un'opzione per limitare l'accesso per proteggere i tuoi contenuti Amazon S3. Quando usi OAI, CloudFront crea un principale con cui Amazon S3 può autenticarsi. I principali autenticati possono accedere ai contenuti in un bucket S3 solo tramite una distribuzione specifica. CloudFront Vedi anche OAC, che fornisce un controllo degli accessi più granulare e avanzato.

O

Vedi la revisione della prontezza operativa.

- NON

Vedi la tecnologia operativa.

VPC in uscita (egress)

In un'architettura AWS multi-account, un VPC che gestisce le connessioni di rete avviate dall'interno di un'applicazione. Nel documento Architettura di riferimento per la sicurezza di AWS si consiglia di configurare l'account di rete con VPC in entrata, in uscita e di ispezione per proteggere l'interfaccia bidirezionale tra l'applicazione e Internet in generale.

P

limite delle autorizzazioni

Una policy di gestione IAM collegata ai principali IAM per impostare le autorizzazioni massime che l'utente o il ruolo possono avere. Per ulteriori informazioni, consulta Limiti delle autorizzazioni nella documentazione di IAM.

informazioni di identificazione personale (PII)

Informazioni che, se visualizzate direttamente o abbinate ad altri dati correlati, possono essere utilizzate per dedurre ragionevolmente l'identità di un individuo. Esempi di informazioni personali includono nomi, indirizzi e informazioni di contatto.

Informazioni che consentono l'identificazione personale degli utenti

Visualizza le informazioni di identificazione personale.

playbook

Una serie di passaggi predefiniti che raccolgono il lavoro associato alle migrazioni, come l'erogazione delle funzioni operative principali nel cloud. Un playbook può assumere la forma di script, runbook automatici o un riepilogo dei processi o dei passaggi necessari per gestire un ambiente modernizzato.

PLC

Vedi controllore logico programmabile.

PLM

Vedi la gestione del ciclo di vita del prodotto.

policy

Un oggetto in grado di definire le autorizzazioni (vedi politica basata sull'identità), specificare le condizioni di accesso (vedi politicabasata sulle risorse) o definire le autorizzazioni massime per tutti gli account di un'organizzazione in (vedi politica di controllo dei servizi). AWS Organizations

persistenza poliglotta

Scelta indipendente della tecnologia di archiviazione di dati di un microservizio in base ai modelli di accesso ai dati e ad altri requisiti. Se i microservizi utilizzano la stessa tecnologia di archiviazione di dati, possono incontrare problemi di implementazione o registrare prestazioni scadenti. I microservizi vengono implementati più facilmente e ottengono prestazioni e scalabilità migliori se utilizzano l'archivio dati più adatto alle loro esigenze. Per ulteriori informazioni, consulta la sezione Abilitazione della persistenza dei dati nei microservizi.

valutazione del portfolio

Un processo di scoperta, analisi e definizione delle priorità del portfolio di applicazioni per pianificare la migrazione. Per ulteriori informazioni, consulta la pagina Valutazione della preparazione alla migrazione.

predicate

Una condizione di interrogazione che restituisce o, in genere, si trova in una clausola`true`. `false` `WHERE`

predicato pushdown

Una tecnica di ottimizzazione delle query del database che filtra i dati della query prima del trasferimento. Ciò riduce la quantità di dati che devono essere recuperati ed elaborati dal database relazionale e migliora le prestazioni delle query.

controllo preventivo

Un controllo di sicurezza progettato per impedire il verificarsi di un evento. Questi controlli sono la prima linea di difesa per impedire accessi non autorizzati o modifiche indesiderate alla rete. Per ulteriori informazioni, consulta Controlli preventivi in Implementazione dei controlli di sicurezza in AWS.

principale

Un'entità in AWS grado di eseguire azioni e accedere alle risorse. Questa entità è in genere un utente root per un Account AWS ruolo IAM o un utente. Per ulteriori informazioni, consulta Principali in Termini e concetti dei ruoli nella documentazione di IAM.

Privacy fin dalla progettazione

Un approccio all'ingegneria dei sistemi che tiene conto della privacy durante l'intero processo di progettazione.

zone ospitate private

Un container che contiene informazioni su come si desidera che Amazon Route 53 risponda alle query DNS per un dominio e i relativi sottodomini all'interno di uno o più VPC. Per ulteriori informazioni, consulta Utilizzo delle zone ospitate private nella documentazione di Route 53.

controllo proattivo

Un controllo di sicurezza progettato per impedire l'implementazione di risorse non conformi. Questi controlli analizzano le risorse prima del loro provisioning. Se la risorsa non è conforme al controllo, non viene fornita. Per ulteriori informazioni, consulta la guida di riferimento sui controlli nella AWS Control Tower documentazione e consulta Controlli proattivi in Implementazione dei controlli di sicurezza su. AWS

gestione del ciclo di vita del prodotto (PLM)

La gestione dei dati e dei processi di un prodotto durante l'intero ciclo di vita, dalla progettazione, sviluppo e lancio, attraverso la crescita e la maturità, fino al declino e alla rimozione.

Ambiente di produzione

Vedi ambiente.

controllore logico programmabile (PLC)

Nella produzione, un computer altamente affidabile e adattabile che monitora le macchine e automatizza i processi di produzione.

pseudonimizzazione

Il processo di sostituzione degli identificatori personali in un set di dati con valori segnaposto. La pseudonimizzazione può aiutare a proteggere la privacy personale. I dati pseudonimizzati sono ancora considerati dati personali.

pubblica/iscriviti (pub/sub)

Un pattern che consente comunicazioni asincrone tra microservizi per migliorare la scalabilità e la reattività. Ad esempio, in un MES basato su microservizi, un microservizio può pubblicare messaggi di eventi su un canale a cui altri microservizi possono abbonarsi. Il sistema può aggiungere nuovi microservizi senza modificare il servizio di pubblicazione.

Q

Piano di query

Una serie di passaggi, come le istruzioni, utilizzati per accedere ai dati in un sistema di database relazionale SQL.

regressione del piano di query

Quando un ottimizzatore del servizio di database sceglie un piano non ottimale rispetto a prima di una determinata modifica all'ambiente di database. Questo può essere causato da modifiche a statistiche, vincoli, impostazioni dell'ambiente, associazioni dei parametri di query e aggiornamenti al motore di database.

R

Matrice RACI

Vedi responsabile, responsabile, consultato, informato (RACI).

ransomware

Un software dannoso progettato per bloccare l'accesso a un sistema informatico o ai dati fino a quando non viene effettuato un pagamento.

Matrice RASCI

Vedi responsabile, responsabile, consultato, informato (RACI).

RCAC

Vedi controllo dell'accesso a righe e colonne.

replica di lettura

Una copia di un database utilizzata per scopi di sola lettura. È possibile indirizzare le query alla replica di lettura per ridurre il carico sul database principale.

riprogettare

Vedi 7 Rs.

obiettivo del punto di ripristino (RPO)

Il periodo di tempo massimo accettabile dall'ultimo punto di ripristino dei dati. Ciò determina quella che viene considerata una perdita di dati accettabile tra l'ultimo punto di ripristino e l'interruzione del servizio.

obiettivo del tempo di ripristino (RTO)

Il ritardo massimo accettabile tra l'interruzione del servizio e il ripristino del servizio.

rifattorizzare

Vedi 7 R.

Regione

Una raccolta di AWS risorse in un'area geografica. Ciascuna Regione AWS è isolata e indipendente dalle altre per fornire tolleranza agli errori, stabilità e resilienza. Per ulteriori informazioni, consulta Specificare cosa può usare Regioni AWS il tuo account.

regressione

Una tecnica di ML che prevede un valore numerico. Ad esempio, per risolvere il problema "A che prezzo verrà venduta questa casa?" un modello di ML potrebbe utilizzare un modello di regressione lineare per prevedere il prezzo di vendita di una casa sulla base di dati noti sulla casa (ad esempio, la metratura).

riospitare

Vedi 7 R.

rilascio

In un processo di implementazione, l'atto di promuovere modifiche a un ambiente di produzione.

trasferisco

Vedi 7 Rs.

ripiattaforma

Vedi 7 Rs.

riacquisto

Vedi 7 Rs.

resilienza

La capacità di un'applicazione di resistere o ripristinare le interruzioni. L'elevata disponibilità e il disaster recovery sono considerazioni comuni quando si pianifica la resilienza in. Cloud AWSPer ulteriori informazioni, vedere Cloud AWS Resilience.

policy basata su risorse

Una policy associata a una risorsa, ad esempio un bucket Amazon S3, un endpoint o una chiave di crittografia. Questo tipo di policy specifica a quali principali è consentito l'accesso, le azioni supportate e qualsiasi altra condizione che deve essere soddisfatta.

matrice di assegnazione di responsabilità (RACI)

Una matrice che definisce i ruoli e le responsabilità di tutte le parti coinvolte nelle attività di migrazione e nelle operazioni cloud. Il nome della matrice deriva dai tipi di responsabilità definiti nella matrice: responsabile (R), responsabile (A), consultato (C) e informato (I). Il tipo di supporto (S) è facoltativo. Se includi il supporto, la matrice viene chiamata matrice RASCI e, se la escludi, viene chiamata matrice RACI.

controllo reattivo

Un controllo di sicurezza progettato per favorire la correzione di eventi avversi o deviazioni dalla baseline di sicurezza. Per ulteriori informazioni, consulta Controlli reattivi in Implementazione dei controlli di sicurezza in AWS.

retain

Vedi 7 R.

andare in pensione

Vedi 7 Rs.

rotazione

Processo di aggiornamento periodico di un segreto per rendere più difficile l'accesso alle credenziali da parte di un utente malintenzionato.

controllo dell'accesso a righe e colonne (RCAC)

L'uso di espressioni SQL di base e flessibili con regole di accesso definite. RCAC è costituito da autorizzazioni di riga e maschere di colonna.

RPO

Vedi l'obiettivo del punto di ripristino.

RTO

Vedi l'obiettivo del tempo di ripristino.

runbook

Un insieme di procedure manuali o automatizzate necessarie per eseguire un'attività specifica. In genere sono progettati per semplificare operazioni o procedure ripetitive con tassi di errore elevati.

S

SAML 2.0

Uno standard aperto utilizzato da molti provider di identità (IdPs). Questa funzionalità abilita il single sign-on (SSO) federato, in modo che gli utenti possano accedere AWS Management Console o chiamare le operazioni AWS API senza che tu debba creare un utente in IAM per tutti i membri dell'organizzazione. Per ulteriori informazioni sulla federazione basata su SAML 2.0, consulta Informazioni sulla federazione basata su SAML 2.0 nella documentazione di IAM.

SCADA

Vedi controllo di supervisione e acquisizione dati.

SCP

Vedi la politica di controllo del servizio.

Secret

In AWS Secrets Manager, informazioni riservate o riservate, come una password o le credenziali utente, archiviate in forma crittografata. È costituito dal valore segreto e dai relativi metadati. Il valore segreto può essere binario, una stringa singola o più stringhe. Per ulteriori informazioni, consulta Secret nella documentazione di Secrets Manager.

controllo di sicurezza

Un guardrail tecnico o amministrativo che impedisce, rileva o riduce la capacità di un autore di minacce di sfruttare una vulnerabilità di sicurezza. Esistono quattro tipi principali di controlli di sicurezza: preventivi, investigativi, reattivi e proattivi.

rafforzamento della sicurezza

Il processo di riduzione della superficie di attacco per renderla più resistente agli attacchi. Può includere azioni come la rimozione di risorse che non sono più necessarie, l'implementazione di best practice di sicurezza che prevedono la concessione del privilegio minimo o la disattivazione di funzionalità non necessarie nei file di configurazione.

sistema di gestione delle informazioni e degli eventi di sicurezza (SIEM)

Strumenti e servizi che combinano sistemi di gestione delle informazioni di sicurezza (SIM) e sistemi di gestione degli eventi di sicurezza (SEM). Un sistema SIEM raccoglie, monitora e analizza i dati da server, reti, dispositivi e altre fonti per rilevare minacce e violazioni della sicurezza e generare avvisi.

automazione della risposta alla sicurezza

Un'azione predefinita e programmata progettata per rispondere o porre rimedio automaticamente a un evento di sicurezza. Queste automazioni fungono da controlli di sicurezza investigativi o reattivi che aiutano a implementare le migliori pratiche di sicurezza. AWS Esempi di azioni di risposta automatizzate includono la modifica di un gruppo di sicurezza VPC, l'applicazione di patch a un'istanza Amazon EC2 o la rotazione delle credenziali.

Crittografia lato server

Crittografia dei dati a destinazione, da parte di chi li riceve. Servizio AWS

Policy di controllo dei servizi (SCP)

Una policy che fornisce il controllo centralizzato sulle autorizzazioni per tutti gli account di un'organizzazione in AWS Organizations. Le SCP definiscono i guardrail o fissano i limiti alle azioni che un amministratore può delegare a utenti o ruoli. Puoi utilizzare le SCP come elenchi consentiti o elenchi di rifiuto, per specificare quali servizi o azioni sono consentiti o proibiti. Per ulteriori informazioni, consulta le politiche di controllo del servizio nella AWS Organizations documentazione.

endpoint del servizio

L'URL del punto di ingresso per un Servizio AWS. Puoi utilizzare l'endpoint per connetterti a livello di programmazione al servizio di destinazione. Per ulteriori informazioni, consulta Endpoint del Servizio AWS nei Riferimenti generali di AWS.

accordo sul livello di servizio (SLA)

Un accordo che chiarisce ciò che un team IT promette di offrire ai propri clienti, ad esempio l'operatività e le prestazioni del servizio.

indicatore del livello di servizio (SLI)

Misurazione di un aspetto prestazionale di un servizio, ad esempio il tasso di errore, la disponibilità o la velocità effettiva.

obiettivo a livello di servizio (SLO)

Una metrica target che rappresenta lo stato di un servizio, misurato da un indicatore del livello di servizio.

Modello di responsabilità condivisa

Un modello che descrive la responsabilità condivisa AWS per la sicurezza e la conformità del cloud. AWS è responsabile della sicurezza del cloud, mentre tu sei responsabile della sicurezza nel cloud. Per ulteriori informazioni, consulta Modello di responsabilità condivisa.

SIEM

Vedi il sistema di gestione delle informazioni e degli eventi sulla sicurezza.

punto di errore singolo (SPOF)

Un guasto in un singolo componente critico di un'applicazione che può disturbare il sistema.

SLAM

Vedi il contratto sul livello di servizio.

SLI

Vedi l'indicatore del livello di servizio.

LENTA

Vedi obiettivo del livello di servizio.

split-and-seed modello

Un modello per dimensionare e accelerare i progetti di modernizzazione. Man mano che vengono definite nuove funzionalità e versioni dei prodotti, il team principale si divide per creare nuovi team di prodotto. Questo aiuta a dimensionare le capacità e i servizi dell'organizzazione, migliora la produttività degli sviluppatori e supporta una rapida innovazione. Per ulteriori informazioni, vedere Approccio graduale alla modernizzazione delle applicazioni in. Cloud AWS

SPOF

Vedi punto di errore singolo.

schema a stella

Una struttura organizzativa di database che utilizza un'unica tabella dei fatti di grandi dimensioni per archiviare i dati transazionali o misurati e utilizza una o più tabelle dimensionali più piccole per memorizzare gli attributi dei dati. Questa struttura è progettata per l'uso in un data warehouse o per scopi di business intelligence.

modello del fico strangolatore

Un approccio alla modernizzazione dei sistemi monolitici mediante la riscrittura e la sostituzione incrementali delle funzionalità del sistema fino alla disattivazione del sistema legacy. Questo modello utilizza l'analogia di una pianta di fico che cresce fino a diventare un albero robusto e alla fine annienta e sostituisce il suo ospite. Il modello è stato introdotto da Martin Fowler come metodo per gestire il rischio durante la riscrittura di sistemi monolitici. Per un esempio di come applicare questo modello, consulta Modernizzazione incrementale dei servizi Web legacy di Microsoft ASP.NET (ASMX) mediante container e Gateway Amazon API.

sottorete

Un intervallo di indirizzi IP nel VPC. Una sottorete deve risiedere in una singola zona di disponibilità.

controllo di supervisione e acquisizione dati (SCADA)

Nella produzione, un sistema che utilizza hardware e software per monitorare gli asset fisici e le operazioni di produzione.

crittografia simmetrica

Un algoritmo di crittografia che utilizza la stessa chiave per crittografare e decrittografare i dati.

test sintetici

Test di un sistema in modo da simulare le interazioni degli utenti per rilevare potenziali problemi o monitorare le prestazioni. Puoi usare Amazon CloudWatch Synthetics per creare questi test.

T

tags

Coppie chiave-valore che fungono da metadati per l'organizzazione delle risorse. AWS Con i tag è possibile a gestire, identificare, organizzare, cercare e filtrare le risorse. Per ulteriori informazioni, consulta Tagging delle risorse AWS.

variabile di destinazione

Il valore che stai cercando di prevedere nel machine learning supervisionato. Questo è indicato anche come variabile di risultato. Ad esempio, in un ambiente di produzione la variabile di destinazione potrebbe essere un difetto del prodotto.

elenco di attività

Uno strumento che viene utilizzato per tenere traccia dei progressi tramite un runbook. Un elenco di attività contiene una panoramica del runbook e un elenco di attività generali da completare. Per ogni attività generale, include la quantità stimata di tempo richiesta, il proprietario e lo stato di avanzamento.

Ambiente di test

Vedi ambiente.

training

Fornire dati da cui trarre ispirazione dal modello di machine learning. I dati di training devono contenere la risposta corretta. L'algoritmo di apprendimento trova nei dati di addestramento i pattern che mappano gli attributi dei dati di input al target (la risposta che si desidera prevedere).

Produce un modello di ML che acquisisce questi modelli. Puoi quindi utilizzare il modello di ML per creare previsioni su nuovi dati di cui non si conosce il target.

Transit Gateway

Un hub di transito di rete che è possibile utilizzare per collegare i VPC e le reti on-premise. Per ulteriori informazioni, consulta Cos'è un gateway di transito nella AWS Transit Gateway documentazione.

flusso di lavoro basato su trunk

Un approccio in cui gli sviluppatori creano e testano le funzionalità localmente in un ramo di funzionalità e quindi uniscono tali modifiche al ramo principale. Il ramo principale viene quindi integrato negli ambienti di sviluppo, preproduzione e produzione, in sequenza.

Accesso attendibile

Concessione delle autorizzazioni a un servizio specificato dall'utente per eseguire attività all'interno dell'organizzazione AWS Organizations e nei suoi account per conto dell'utente. Il servizio attendibile crea un ruolo collegato al servizio in ogni account, quando tale ruolo è necessario, per eseguire attività di gestione per conto dell'utente. Per ulteriori informazioni, consulta Utilizzo AWS Organizations con altri AWS servizi nella AWS Organizations documentazione.

regolazione

Modificare alcuni aspetti del processo di training per migliorare la precisione del modello di ML. Ad esempio, puoi addestrare il modello di ML generando un set di etichette, aggiungendo etichette e quindi ripetendo questi passaggi più volte con impostazioni diverse per ottimizzare il modello.

team da due pizze

Una piccola DevOps squadra che puoi sfamare con due pizze. Un team composto da due persone garantisce la migliore opportunità possibile di collaborazione nello sviluppo del software.

U

incertezza

Un concetto che si riferisce a informazioni imprecise, incomplete o sconosciute che possono minare l'affidabilità dei modelli di machine learning predittivi. Esistono due tipi di incertezza: l'incertezza epistemica, che è causata da dati limitati e incompleti, mentre l'incertezza aleatoria

è causata dal rumore e dalla casualità insiti nei dati. Per ulteriori informazioni, consulta la guida Quantificazione dell'incertezza nei sistemi di deep learning.

compiti indifferenziati

Conosciuto anche come sollevamento di carichi pesanti, è un lavoro necessario per creare e far funzionare un'applicazione, ma che non apporta valore diretto all'utente finale né offre vantaggi competitivi. Esempi di attività indifferenziate includono l'approvvigionamento, la manutenzione e la pianificazione della capacità.

ambienti superiori

Vedi ambiente.

V

vacuum

Un'operazione di manutenzione del database che prevede la pulizia dopo aggiornamenti incrementali per recuperare lo spazio di archiviazione e migliorare le prestazioni.

controllo delle versioni

Processi e strumenti che tengono traccia delle modifiche, ad esempio le modifiche al codice di origine in un repository.

Peering VPC

Una connessione tra due VPC che consente di instradare il traffico tramite indirizzi IP privati. Per ulteriori informazioni, consulta Che cos'è il peering VPC? nella documentazione di Amazon VPC.

vulnerabilità

Un difetto software o hardware che compromette la sicurezza del sistema.

W

cache calda

Una cache del buffer che contiene dati correnti e pertinenti a cui si accede frequentemente. L'istanza di database può leggere dalla cache del buffer, il che richiede meno tempo rispetto alla lettura dalla memoria dal disco principale.

dati caldi

Dati a cui si accede raramente. Quando si eseguono interrogazioni di questo tipo di dati, in genere sono accettabili query moderatamente lente.

funzione finestra

Una funzione SQL che esegue un calcolo su un gruppo di righe che si riferiscono in qualche modo al record corrente. Le funzioni della finestra sono utili per l'elaborazione di attività, come il calcolo di una media mobile o l'accesso al valore delle righe in base alla posizione relativa della riga corrente.

Carico di lavoro

Una raccolta di risorse e codice che fornisce valore aziendale, ad esempio un'applicazione rivolta ai clienti o un processo back-end.

flusso di lavoro

Gruppi funzionali in un progetto di migrazione responsabili di una serie specifica di attività. Ogni flusso di lavoro è indipendente ma supporta gli altri flussi di lavoro del progetto. Ad esempio, il flusso di lavoro del portfolio è responsabile della definizione delle priorità delle applicazioni, della pianificazione delle ondate e della raccolta dei metadati di migrazione. Il flusso di lavoro del portfolio fornisce queste risorse al flusso di lavoro di migrazione, che quindi migra i server e le applicazioni.

VERME

Vedi scrivere una volta, leggere molti.

WQF

Vedi AWS Workload Qualification Framework.

scrivi una volta, leggi molte (WORM)

Un modello di storage che scrive i dati una sola volta e ne impedisce l'eliminazione o la modifica. Gli utenti autorizzati possono leggere i dati tutte le volte che è necessario, ma non possono modificarli. Questa infrastruttura di archiviazione dei dati è considerata immutabile.

Z

exploit zero-day

Un attacco, in genere malware, che sfrutta una vulnerabilità zero-day.

vulnerabilità zero-day

Un difetto o una vulnerabilità assoluta in un sistema di produzione. Gli autori delle minacce possono utilizzare questo tipo di vulnerabilità per attaccare il sistema. Gli sviluppatori vengono spesso a conoscenza della vulnerabilità causata dall'attacco.

applicazione zombie

Un'applicazione che prevede un utilizzo CPU e memoria inferiore al 5%. In un progetto di migrazione, è normale ritirare queste applicazioni.

Le traduzioni sono generate tramite traduzione automatica. In caso di conflitto tra il contenuto di una traduzione e la versione originale in Inglese, quest'ultima prevarrà.