Home > Architecting

Strategie di Decoupling: single-queue o una coda per microservizio?

29 Marzo 2024 - 8 min. read

Amazon Simple Queue Service (SQS) ***Decoupled Architectures*** ***Microservices***

Perché disaccoppiare i processi?

Il disaccoppiamento dei carichi di lavoro complessi è una pratica che le persone talvolta intraprendono senza una chiara comprensione dei vantaggi e dei rischi.

Sentiamo spesso parlare di applicazioni modulari, costruite con microservizi, dove è possibile decidere cosa attivare o disattivare senza alcun impatto sulle altre parti, o che possono scalare all'infinito.

Tuttavia, quali sono le pratiche da utilizzare per il monitoraggio? Esiste il rischio di finire per lavorare in un ambiente disordinato che, per il bene dello scaling e di tutti gli altri vantaggi, rovina l'esperienza dello sviluppatore? Seguendo questo link troverete un ottimo articolo scritto dal mio collega e amico, Damiano, che approfondisce le considerazioni che tutti dovrebbero fare quando lavorano con i microservizi.

Oggi discuteremo due strategie di decoupling, dei loro benefici, delle loro sfide e anche di quando una è migliore dell'altra. Il focus sarà sui servizi AWS perché ci sono molte opzioni (Amazon SQS, EventBridge, SNS), ma queste considerazioni possono essere applicate anche ad altri strumenti.

Strategia 1: approccio a coda singola

Il nome è piuttosto autoesplicativo: il nostro carico di lavoro ha un unico punto di

ingresso per lo scambio e la lettura dei messaggi. Ogni produttore di eventi invia il suo payload a quella coda, dove sono in ascolto più consumatori. Questo approccio viene spesso respinto come troppo caotico poiché la coda contiene messaggi non correlati e non si allinea con una mentalità "orientata ai microservizi".

Anche se questo potrebbe essere vero: i vantaggi esistono e non dovrebbero essere trascurati.

Immagina di lavorare su un'applicazione in cui abbiamo molti produttori di eventi e solo alcuni processi che elaborano quei messaggi. Quando il numero di tipi di eventi è piccolo, un approccio a coda singola è davvero facile da implementare, monitorare ed estendere. Ogni volta che viene creato un nuovo produttore, esso dovrà semplicemente inviare messaggi all'unica coda esistente. D'altra parte, i consumatori dovranno filtrare i messaggi per tipo per gestire solo gli eventi all'interno della propria competenza, il che richiede un po' più di logica da implementare.

Questo è un approccio molto più semplice per la costruzione di un'applicazione disaccoppiata, ma, come detto in apertura di questo articolo, non dovremmo mai iniziare con la complessità.

Ciò che è determinante è la consapevolezza dei picchi di traffico a cui la coda potrebbe essere sottoposta; avere troppi messaggi che passano attraverso lo stesso tunnel potrebbe creare congestione.

Strategia 2: approccio con coda dedicata ai microservizi

Mentre l'approccio a coda singola è ideale per carichi di lavoro ridotti, un'applicazione più complessa con più persone al lavoro è più facile da mantenere se ogni microservizio ha la sua coda dedicata. Ciò consente un monitoraggio più dettagliato poiché può essere effettuato a livello di microservizio e i messaggi che attraversano la coda seguiranno uno standard. Questa strategia consente anche di creare una gestione più dettagliata delle autorizzazioni: se desideri stabilire una comunicazione tra due microservizi, dovrai concedere l'accesso al produttore per generare eventi all'interno della coda del consumatore.

Sii consapevole che, **ogni volta che uno dei tuoi microservizi genera eventi per un**

altro microservizio, nasce una dipendenza, creando quindi un antipattern.

I microservizi dovrebbero essere indipendenti l'uno dall'altro, dovrebbero avere il proprio database, le proprie risorse computazionali e non dovrebbero comunicare tra loro. Questa è la teoria naturalmente; nel mondo reale dobbiamo analizzare ogni singolo caso per capire se attenerci ai modelli comuni, o se deviare da uno standard può aiutarci a creare flussi di lavoro più facili da capire.

In ogni caso, una buona strategia per gestire flussi di lavoro complessi in cui è necessario chiamare più microservizi può essere quella di impostare **macchine a stati** che gestiscano in modo controllato l'inoltro degli eventi, l'analisi degli output e la presa di decisioni. Ciò rende la nostra applicazione facile da debuggare, monitorare ed estendere con nuovi microservizi (in quanto è sufficiente che seguano lo standard definito). Se rendi i tuoi microservizi davvero indipendenti l'uno dall'altro, sarai anche in grado di testarli senza creare effetti collaterali.

Parliamo di monitoraggio

Abbiamo già citato il tema del monitoraggio due volte in questo articolo. È il momento di analizzarlo meglio. Quando si lavora con i microservizi, di solito si creano comunicazioni asincrone tra di essi ed è spesso difficile tenere traccia del ciclo di vita di un evento specifico: se qualcosa va storto, dobbiamo essere in grado di capire quale processo è fallito, come dovremmo rimediare all'errore e come possiamo prevenire effetti collaterali che possono incidere sull'esperienza dell'utente. Questo è un argomento che i DevOps devono analizzare approfonditamente nella prima fase del progetto.

Su AWS, strumenti come le **dashboard di CloudWatch** possono essere davvero utili per ottenere informazioni su come si comporta il carico di lavoro, ma bisogna andare oltre.

Naturalmente, avere la possibilità di monitorare il numero di messaggi all'interno della coda o il numero di eventi che non sono riusciti a essere elaborati è importante per capire dove si trovano i *single-point of failure* e dove dovremmo considerare lo scaling, ma abbiamo comunque bisogno di qualcosa che ci aiuti a monitorare il carico di lavoro a livello di applicazione.

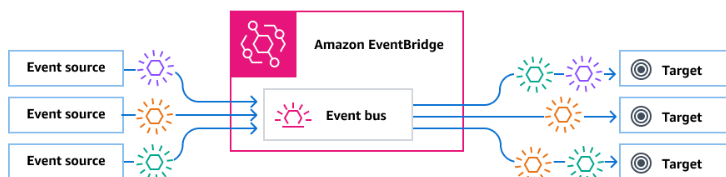
Ecco dove AWS X-Ray può aiutarci: AWS X-Ray ci consente di tracciare e analizzare le

richieste mentre viaggiano attraverso la nostra applicazione. Questo ci fornisce una visione dettagliata del suo comportamento, consentendoci di identificare dove si verificano i colli di bottiglia e quali servizi causano ritardi. Questa visione end-to-end del flusso di lavoro ci consente di identificare e risolvere i problemi in modo più efficace. Con X-Ray possiamo ottenere metriche utili come il tempo impiegato dal nostro microservizio per fare qualcosa (come una richiesta HTTP) o quali altri microservizi sono stati chiamati per elaborare un evento.

Servizi AWS per il disaccoppiamento

Quando parliamo del disaccoppiamento dei processi, il primo servizio AWS che ci viene in mente è Amazon SQS. È stato creato proprio con questo scopo: i produttori possono inserire i loro eventi all'interno di una coda mentre i consumatori effettuano il polling su quella coda per vedere se ci sono dei messaggi da elaborare. Amazon SQS offre diverse funzionalità che possono essere utili per il nostro obiettivo, come le *dead-letter queue* per i messaggi non elaborati, *long-polling* per evitare sul lato del consumatore di inondare la coda con richieste che non produrranno nessun output, e code FIFO se abbiamo bisogno di elaborare i nostri eventi nell'ordine in cui sono stati prodotti.

Amazon SQS è un ottimo servizio, ma possiamo trovare in altri servizi AWS delle funzionalità che possono soddisfare meglio le nostre esigenze. Un servizio AWS che mi piace particolarmente è Amazon EventBridge. Con Amazon EventBridge possiamo registrare più destinazioni per essere notificati ogni volta che viene pubblicato un messaggio in un bus degli eventi, ma possiamo anche filtrare questi messaggi specificando dei pattern per inoltrarli alla destinazione corretta. Questo è davvero utile se stiamo costruendo un approccio a coda singola!



Sorgente immagine: AWS Documentation

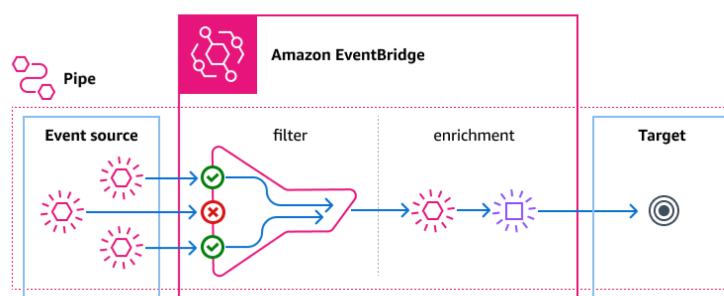
Ci sono altre due funzionalità di Amazon EventBridge che adoro: *archive/replay* e pipes.

Gli **archivi** sono spazi in cui possiamo memorizzare eventi che rispecchiano un pattern,

hanno un periodo di conservazione e possono essere riprodotti successivamente.

La **riproduzione** degli eventi ci aiuta a testare i nostri microservizi per assicurarci di non aver aggiunto alcuna regressione mentre lavoriamo su una nuova funzionalità, ma possiamo anche rielaborare eventi precedentemente falliti. Gli eventi vengono riprodotti nell'ordine esatto in cui sono stati ricevuti inizialmente.

EventBridge **Pipes** è una funzionalità che è stata rilasciata a dicembre 2022 e ci aiuta a creare una piccola logica di business per filtrare ulteriormente i nostri eventi o arricchirli con altre informazioni. La funzione di arricchimento può essere realizzata facilmente con una Lambda function, ma anche con una funzione di passaggio o una richiesta HTTP. Penso che Lambda sia il servizio più adatto a questo scopo, poiché desidero che l'arricchimento sia il più veloce possibile. Le pipes hanno un costo e dovrebbero essere prese in considerazione nella pianificazione dell'architettura, specialmente quando ci aspettiamo che tonnellate di eventi da elaborare!



Sorgente immagine: AWS Documentation

Un esempio concreto

Recentemente ho utilizzato un modello di disaccoppiamento in un progetto in cui dovevo costruire un'API REST che, quando chiamata, doveva eseguire alcune logiche e query sul database e, a seconda del risultato della query, effettuare alcune chiamate API a servizi esterni che implementano protocolli diversi.

Quello che ho fatto è stato semplicemente suddividere l'intera logica in due macro categorie, ognuna di proprietà di una funzione Lambda. La prima funzione Lambda elabora il payload ricevuto tramite la chiamata API, effettua le query sul database e recupera tutti i parametri necessari per gli step successivi. Infine crea un payload che invia a un bus di EventBridge, dove tra i parametri c'è anche una proprietà che specifica il tipo di servizio di terze parti che deve essere chiamato. Poi ho creato un set di funzioni

Lambda, una per ogni servizio di terze parti, tutte registrate come target sullo stesso bus con un pattern che corrisponde solo al loro tipo di servizio. Queste funzioni si occupano di gestire la logica di business specifica per quel servizio di terze parti

In questo modo sono stato in grado di creare molte funzioni Lambda più piccole e facili da mantenere che si occupano solo di ciò che devono fare, con una funzione Lambda produttore che pubblica sempre messaggi sullo stesso servizio AWS e un set di funzioni Lambda consumatrici, tutte simili per quanto riguarda la configurazione generale ma che differiscono per la logica di business.

Per riassumere...

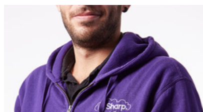
I pattern sono stati creati per semplificarci la vita e non reinventare la ruota ogni volta. Sono uno standard ben noto che ogni DevOps con un po' di esperienza dovrebbe conoscere perché mirano a risolvere problemi reali.

Allo stesso tempo, dobbiamo essere concentrati sul nostro obiettivo: penso sempre che il miglior approccio sia iniziare in modo semplice e rendere le cose complesse in seguito, se necessario. In passato ho utilizzato entrambe le strategie, scegliendo una piuttosto che l'altra in base alla complessità e alla criticità del progetto su cui stavo lavorando.

Qual è la vostra esperienza su questo argomento? Come approcciate il tema del decoupling? Fatecelo sapere nei commenti!

About Proud2beCloud

Proud2beCloud è il blog di beSharp, APN Premier Consulting Partner italiano esperto nella progettazione, implementazione e gestione di infrastrutture Cloud complesse e servizi AWS avanzati. Prima di essere scrittori, siamo Solutions Architect che, dal 2007, lavorano quotidianamente con i servizi AWS. Siamo innovatori alla costante ricerca della soluzione più all'avanguardia per noi e per i nostri clienti. Su Proud2beCloud condividiamo regolarmente i nostri migliori spunti con chi come noi, per lavoro o per passione, lavora con il Cloud di AWS. Partecipa alla discussione!



Mattia Costamagna

Ingegnere DevOps e sviluppatore cloud-native @ beSharp. Adoro passare il mio tempo libero a leggere romanzi e ascoltare musica rock e blues degli anni '70. Sempre alla ricerca di nuove tecnologie e framework da testare e utilizzare. La birra artigianale è il mio carburante!

Copyright © 2011-2024 by beSharp spa - P.IVA IT02415160189