Architetture Event-Driven a KM0: dal producer al consumer

30 Luglio 2025 - 9 min. read

Event-Driven Architecture (EDA)

Introduzione

In questo primo di due articoli, approfondiamo un argomento noto da tempo ma recentemente al centro di un interesse crescente: le Architetture Event-Driven ed Event-Based (EDA).

Nonostante se ne parli sempre più spesso, questo non significa che l'argomento sia diventato più chiaro. L'obiettivo di questi articoli è offrire una spiegazione semplice e comprensibile dei principi fondamentali alla base di questi paradigmi architetturali.

Al centro delle EDA si trovano due ruoli principali: il Producer e il Consumer. Il Producer ha il compito di generare e inviare un messaggio. Dall'altro lato, il Consumer riceve questo messaggio e reagisce, tipicamente eseguendo una logica di business o attivando un flusso di lavoro. Questa interazione costituisce la spina dorsale del flusso informativo nelle EDA.

La semantica degli eventi

Un messaggio può rappresentare diversi tipi di comunicazione a seconda dell'intento del Producer. Più comunemente, i messaggi si dividono in due categorie: comandi ed eventi.

Un comando viene utilizzato quando il Producer desidera richiedere un'azione specifica al Consumer. I comandi sono tipicamente imperativi e si aspettano un certo risultato o effetto collaterale.

Al contrario, un evento serve per notificare qualcosa che è accaduto — spesso rappresentando un cambiamento di stato o il verificarsi di un evento di business rilevante. A differenza dei comandi, gli eventi non si aspettano una risposta diretta o un'azione da parte del Consumer.

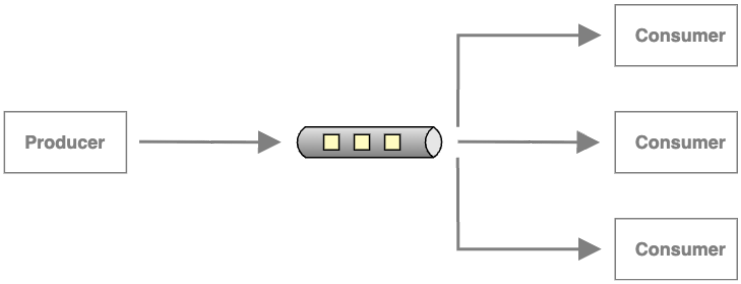
Questa distinzione tra comandi ed eventi è fondamentale per comprendere come comunicano i componenti nelle EDA e influenza profondamente il modo in cui vengono progettati i sistemi.

Canali di messaggistica

Un Producer sfrutta canali di messaggistica per inoltrare i messaggi verso i Consumer. Come osserva Gregor Hohpe in uno dei suoi "ramblings", le EDAs sono comunemente associate ai canali Publish-Subscribe (Pub/Sub), poiché più destinatari possono essere interessati a reagire ad un singolo evento. Questo si contrappone ai canali Point-to-Point, che sono tipicamente usati per inviare comandi o documenti a un singolo Consumer.

Nell'ecosistema AWS, questi due tipi di canali sono ben rappresentati: le code SQS rappresentano i canali Point-to-Point, mentre i topic SNS sono una chiara implementazione di canali Publish-Subscribe. Con le code SQS, ogni messaggio è indirizzato a un solo Consumer — solitamente un worker o un'applicazione che estrae messaggi dalla coda — garantendo che venga processato una sola volta. Al contrario, i topic SNS permettono che un singolo messaggio venga inviato a più consumatori, detti subscriber, che si sono sottoscritti al topic. Questo modello consente una comunicazione decentrata, in cui gli eventi possono propagarsi simultaneamente a diversi sistemi, supportando scalabilità ed estensibilità.

Canale di messaggistica Point-to-Point



Canale di messaggistica Publish-Subscribe

Enterprise Integration Patterns di Gregor Hohpe fornisce definizioni canoniche di entrambi i modelli, e queste idee sono fondamentali per le EDA moderne.

Le dimensioni del coupling

Quando colleghiamo un Producer ad un Consumer tramite un canale di messaggistica, introduciamo un certo tipo di coupling. Una delle promesse principali delle EDA è massimizzare la variabilità indipendente di Producer e Consumer. Ciò significa che i componenti (in particolare i Consumer) dovrebbero poter cambiare senza impattare il resto del sistema.

Ma il coupling non è binario; esistono diverse dimensioni, ciascuna delle quali influisce su un aspetto diverso del comportamento del sistema. Esploriamo le cinque dimensioni principali del coupling — e come i diversi tipi di canali di messaggistica le supportano o le ostacolano.

Temporal coupling

Cosa rappresenta: dipendenza temporale tra Producer e Consumer.

Modifiche tipiche del Consumer: diventa temporaneamente non disponibile, rallenta a causa della latenza o del carico.

Nei sistemi sincroni e fortemente accoppiati, un Producer deve attendere la risposta del Consumer. Ciò crea fragilità: se il Consumer non è disponibile, il Producer rimane bloccato.

Supporto dei canali di messaggi: sia i canali Point-to-Point (es. SQS) sia quelli Pub/Sub (es. SNS) disaccoppiano i componenti attraverso la messaggistica asincrona.

Il Producer emette un messaggio e prosegue. I consumatori prelevano o ricevono i messaggi quando sono pronti.

Questo disaccoppiamento migliora la resilienza del sistema, poiché i produttori non dipendono dalla disponibilità immediata dei consumatori.

Location Coupling

Cosa rappresenta: dipendenza del *Producer* dalla posizione fisica o logica del *Consumer*.

Modifiche tipiche del *Consumer*: viene spostato su un altro host, in una availability zone diversa, in una nuova regione o viene scalato orizzontalmente su più istanze.

In scenari tradizionali come le chiamate RPC, il *Producer* è consapevole dell'indirizzo del *Consumer*, che si tratti di un hostname o di un endpoint. Questo implica che ogni cambiamento infrastrutturale — come la migrazione o la replica del *Consumer* — può richiedere modifiche al *Producer*.

Supporto dei canali di messaggi:

I canali di messaggistica (es. SQS, SNS, EventBridge) riducono questo tipo di accoppiamento, poiché sono generalmente costrutti logici. Il *Producer* invia messaggi a un canale, senza conoscere la posizione dei *Consumer*. Questo permette ai *Consumer* di spostarsi, duplicarsi o scalare senza impattare i *Producer*.

Tuttavia, quando un canale è vincolato a una specifica availability zone o regione, il disaccoppiamento geografico può essere parziale.

Space coupling

Cosa rappresenta: una dipendenza strutturale che presuppone una comunicazione diretta tra *Producer* e *Consumer*, senza componenti intermedi.

Modifiche tipiche del *Consumer*: il Consumer viene spostato dietro un proxy, un API gateway, o un altro intermediario.

Lo *space coupling* si riferisce alla difficoltà di introdurre un intermediario tra *Producer* e *Consumer* senza dover modificare la logica del mittente. In architetture fortemente accoppiate, il *Producer* si aspetta di comunicare direttamente con il *Consumer*, e qualsiasi cambiamento nella catena — come l'inserimento di un layer di routing, filtraggio o trasformazione — richiede spesso aggiornamenti sul lato del *Producer* stesso.

Supporto dei canali di messaggi:

I canali di messaggistica — sia *Point-to-Point* che *Publish-Subscribe* — offrono un valido disaccoppiamento da questo punto di vista. Partendo da una situazione in cui il

Producer comunica con il Consumer tramite un canale di messaggistica, è possibile introdurre broker, *event bus*, o altri componenti intermedi senza alterare il comportamento del *Producer*.

Topology coupling

Cos'è: la possibilità di aggiungere nuovi Consumer (nuove logiche applicative) senza impattare il Producer o introdurre anti-pattern.

Modifiche tipiche del Consumer: Una nuova applicazione Consumer ha bisogno di ricevere gli stessi eventi.

Qui la distinzione tra tipi di canali diventa critica.

Nei canali Point-to-Point come SQS, solo un Consumer può ricevere con successo ogni messaggio. Aggiungere un secondo Consumer alla stessa coda è un anti-pattern: si creano race condition, dove solo uno di n Consumer riceve il messaggio.

Al contrario, i canali Publish-Subscribe come i topic SNS sono progettati per supportare più consumatori. Ogni subscriber riceve una copia del messaggio.

Implicazioni dei canali di messaggi:

- **Point-to-Point:** non adatto a topologie dinamiche che prevedono l'aggiunta di nuove logiche Consumer.
- **Pub/Sub:** Ideale per architetture estensibili dove si possono aggiungere nuovi Consumer senza modificare il Producer.

Questi modelli tradizionali rispondono a molti casi d'uso; possono, però, risultare limitanti quando si desidera consegnare un messaggio (comando o evento) ad un gruppo specifico di Consumer — senza impattare il Producer o l'infrastruttura esistente.

Torneremo su questo punto a breve!

Coupling di Formato e Semantico

Cos'è: Dipendenza tra Producer e Consumer a livello di struttura e significato dei dati scambiati.

Modifiche tipiche del Consumer: Evoluzione dello schema (es. aggiunta, rimozione o rinomina di campi), cambiamenti nel significato o nel formato dei valori attesi.

Anche in sistemi ben disaccoppiati nel tempo o nella topologia, il legame tra Producer e Consumer può persistere a livello di dati. Questo tipo di accoppiamento si manifesta in due forme principali:

- **Coupling di formato:** Il Consumer si aspetta una struttura precisa — nomi, ordine, o tipi dei campi — e può fallire se questa cambia. Formati rigidi come i record EDI possono rompersi anche per una semplice riorganizzazione di campi. Al contrario, formati con metadati espliciti come JSON o XML tendono a essere più tolleranti.
- **Coupling semantico:** Anche se la struttura resta invariata, il significato attribuito a un campo può cambiare. Per esempio, un campo `"status": "open"` potrebbe assumere nuovi significati nel tempo, con impatti sui Consumer che ne fanno uso implicito.

Implicazioni per l'architettura:

È difficile, se non impossibile, disaccoppiare completamente Producer e Consumer dal punto di vista del formato e della semantica dei dati.

Producer e Consumer devono accordarsi su un contratto che comprende la definizione dei campi (obbligatori e non) inclusi in tutte le richieste ed in tutte le risposte. In questo modo è possibile descrivere il formato e la semantica dei messaggi che vengono scambiati tra Producer e Consumer.

Per rendere questo contratto più aperto ai cambiamenti, è fondamentale adottare:

- Formati auto-descrittivi (facilmente interpretabili) come JSON, XML o Protocol Buffers
- Tecniche di versionamento e uso di campi opzionali
- Trasformazioni lato consumer o tramite componenti intermedi

Questo tipo di coupling, se trascurato, può facilmente introdurre rotture anche in sistemi che sembrano disaccoppiati su altri piani.

Riepilogo delle dimensioni del coupling

Ogni dimensione del coupling riflette un diverso tipo di dipendenza tra componenti. I canali di messaggistica come SQS e SNS aiutano a disaccoppiare produttori e consumatori su diverse dimensioni — ma non sono ideali per tutti i casi d'uso.

Coupling	Change	Pt-to-Pt	Pub/Sub
Temporal	Availability, latency	+	+
Location	Move, scale-out	+	+
Space	Insert intermediary	+	+
Topology	Add recipient	-	+
Format	Schema change	-	-
Semantic	Field meaning	-	-

Dove i canali Point-to-Point risultano limitanti, soprattutto nella flessibilità topologica, strumenti come Amazon EventBridge offrono un modello più dinamico e scalabile. EventBridge è un event bus Serverless che consente di instradare eventi verso uno o più Consumer in base a regole flessibili e basate sul contenuto. A differenza della messaggistica tradizionale Point-to-Point, dove aggiungere un nuovo Consumer può introdurre race condition o richiedere cambi architetturali, EventBridge disaccoppia la logica di routing dal Producer. Questo significa che è possibile indirizzare lo stesso evento a più Consumer, o solo a uno selezionato, senza modificare il codice upstream o duplicare i messaggi.

Supportando sia i pattern Publish-Subscribe che quelli Point-to-Point da un unico flusso di eventi, EventBridge colma le lacune lasciate dagli strumenti di messaggistica convenzionali. Con funzionalità integrate di filtering, delivery mirato e routing molti-a-molti, consente di costruire sistemi più facili da evolvere.

Event-Driven o Event-Based?

A questo punto, dovresti avere una buona comprensione di alto livello di cosa significhi per un sistema lavorare con eventi, il che sarà utile quando passeremo al lato pratico. Nel panorama software attuale, i termini **event-driven** ed **event-based** vengono spesso usati in modo intercambiabile, ma in realtà si riferiscono a prospettive differenti. A un livello generale, un **evento** è semplicemente una notifica che qualcosa è accaduto — un fatto immutabile che non può essere annullato.

La differenza chiave risiede nel tipo di eventi a cui i sistemi reagiscono. I sistemi **event-based** trattano principalmente *eventi tecnici*, come il caricamento di un file o l'attivazione di un timer. I sistemi **event-driven**, invece, si concentrano su *eventi di business* — accadimenti significativi nel dominio, come la registrazione di un nuovo cliente o la creazione di un ordine. Questa distinzione influisce sul modo in cui progettiamo, costruiamo e comunichiamo sui nostri sistemi, assicurandoci che il software reagisca al cambiamento in un modo che sia allineato con gli obiettivi del business.

Conclusione

In questa prima tappa del nostro viaggio nel mondo delle EDA abbiamo messo le basi, esplorando insieme i concetti chiave: dalla doppia natura dei messaggi (comandi ed eventi), al ruolo centrale dei canali di messaggistica nei vari pattern di comunicazione, fino a capire le diverse sfumature del coupling.

Capire bene questi principi ti dà gli strumenti per andare oltre le mode e le parole d'ordine, e scoprire cosa davvero rende i sistemi event-based ed event-driven così potenti. Sono i mattoni con cui costruire architetture flessibili, scalabili e resilienti, pronte a crescere con il tuo business.

E questo è solo l'inizio.

Nel prossimo articolo ci metteremo all'opera e vedremo come creare un event router su AWS con EventBridge: impareremo a usare regole di routing basate sul contenuto degli eventi per indirizzarli solo ai consumer giusti, e vedremo insieme come gestire errori, monitorare il sistema e mantenerlo sicuro e pronto all'evoluzione.

Restate sintonizzati!

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!

Eric Villa

Solutions Architect @beSharp | AWS Community Builder | AWS Authorized Instructor