# Un tuffo nelle architetture cloud disaccoppiate con gli Event Bus

2 Febbraio 2024 - 11 min. read

Decoupled Architectures | Event Buses | Microservices

Per trasformarsi ed essere all'avanguardia nel mondo del Cloud Computing le aziende devono assicurarsi scalabilità, resilienza e flessibilità dei loro sistemi.

Per far ciò, sempre più organizzazioni stanno migrando da architetture monolitiche a sistemi distribuiti.

Nel passaggio da applicazioni monolitiche a sistemi distribuiti, padroneggiare le strategie e i pattern di disaccoppiamento diventa cruciale.

Il disaccoppiamento (decoupling) è un cambiamento strategico che divide i componenti all'interno di un
sistema, facendoli funzionare in modo indipendente e mantenendo l'interoperabilità.

Questo articolo è ispirato da una sessione dell'AWS re:Invent 2023 Dal titolo "Advanced integration patterns & trade-offs for loosely coupled systems"

In questo articolo esploreremo le architetture cloud disaccoppiate concentrandoci sulla funzione
degli event bus. Gli event bus fungono da base per il coordinamento e la comunicazione delle applicazioni in un sistema distribuito.

Questo articolo spiegherà le complessità della moderna architettura cloud ai tecnici veterani e ai
leader aziendali che desiderano comprenderne le basi. Evidenzierà, inoltre, i vantaggi concreti
dell'adozione di un approccio disaccoppiato.

## Da un'applicazione monolitica, ai microservizi: il disaccoppiamento nel contesto del cloud

Il processo di scomposizione dell'applicazione monolitica in microservizi può sembrare molto impegnativo all'inizio. Comporta però diversi vantaggi che possono essere riassunti come
"**indipendenza dai componenti**".

Per citare solo alcuni di questi:

- **Scalabilità**: i componenti possono fare scale in/out a seconda del loro utilizzo in modo indipendente.
  Questo di solito si traduce in un migliore utilizzo delle risorse e in una riduzione dei costi ed è anche un fattore-chiave per l'alta disponibilità dell'intero sistema.
- **Resilienza:** è meno probabile che un guasto in un singolo microservizio si propaghi all'intero sistema creando un fallimento grave.
- **Manutenibilità**: i microservizi, come suggerisce il nome, sono solitamente componenti più piccoli. In termini di code base, la manutenzione e l'aggiornamento diventano quindi più semplici, con conseguenti aggiornamenti più rapidi e consistenti. Inoltre, i microservizi possono essere anche indipendenti dal punto di vista del linguaggio di programmazione e/o della tecnologia usata, quindi il loro sviluppo può essere suddiviso e parallelizzato tra più team. Questo può aumentare enormemente la velocità di sviluppo del prodotto e delle relative funzionalità.
- **Facilità di test:** un altro aspetto della code base ridotta dei microservizi è che il test del codice diventa più semplice. Gli sviluppatori dovranno scrivere test unitari solo per lo specifico microservizio. Questo solitamente si traduce in un maggior numero di test scritti e in una migliore copertura del codice che contribuisce alla robustezza del software.

Questa indipendenza è fondamentale nell'ambiente dinamico ed elastico del cloud, dove l'agilità, unita alla scalabilità fornita dai servizi AWS, aiuta realmente a sviluppare e migliorare
rapidamente l'applicazione.

## Componenti a basso accoppiamento

Un modello di progettazione comune è quello dei "componenti a basso accoppiamento": i componenti devono essere progettati per operare indipendentemente l'uno dall'altro, interagendo con una dipendenza minima o nulla.

In questo senso, i componenti hanno una conoscenza minima l'uno dell'altro; sanno solo come scambiare informazioni tra loro attraverso interfacce ben definite o contratti che standardizzano la comunicazione: strutture di messaggi, formato dei dati e così via.

Come accennato in precedenza, l'accoppiamento può essere visto sotto molti aspetti diversi: dall'accoppiamento tecnologico, in cui le applicazioni condividono lo stesso linguaggio di programmazione o la stessa tecnologia, all'accoppiamento della comunicazione e della conversazione: la comunicazione è sincrona o asincrona? Come comunichiamo: risultati completi, paginazione, caching? E il meccanismo di riprova?

Dato che lo scopo di questo articolo è incentrato sulla comunicazione tra sistemi, ci concentreremo sulle strategie di disaccoppiamento correlate.

## Gli Event Bus: la spina dorsale delle architetture distribuite nel Cloud

Parlando di comunicazione tra sistemi, gli event bus sono la soluzione perfetta per ottenere il disaccoppiamento.

Nelle architetture cloud, i sistemi comunicano e si integrano tra loro utilizzando gli eventi. Gli event bus possono essere utilizzati come sistema di interconnessione tra produttori e consumatori di eventi, orchestrando il flusso di eventi tra le parti.

La caratteristica principale è che, poiché il bus si trova nel mezzo tra produttori e consumatori, è possibile collegare più sistemi in una relazione *molti-a-molti* con il minimo sforzo: i produttori invieranno i loro eventi al bus dedicato e tutti i consumatori interessati a un evento specifico lo riceveranno ed elaboreranno.

A titolo di esempio, si pensi a un sito di e-commerce che deve elaborare un ordine sotto diversi aspetti: elaborazione del pagamento, gestione dell'inventario e spedizione dell'ordine.

Lo stesso vale per i consumatori che possono essere interessati allo stesso tipo di evento, prodotto da più sistemi. Un esempio di ciò può essere un sistema di notifiche comune per l'intera architettura.

In questo senso, si può notare come l'uso degli event bus è simile a quello di un sistema Pub/Sub.

## Gestione degli event bus su AWS: Amazon EventBridge

Il servizio AWS che consente di creare e utilizzare gli event bus è **Amazon EventBridge**. Per ogni account AWS è previsto un event bus pre-creato chiamato "**default**". Questo bus viene

utilizzato da tutti i servizi AWS all'interno dell'account per pubblicare tutti gli eventi relativi alle loro operazioni. Questo bus è molto importante perché è la spina dorsale di tutta l'architettura event-driven di AWS. (Un rapido esempio: Funzione Lambda attivata su oggetti caricati su Amazon S3).

Oltre all'event bus predefinito, **è possibile creare event bus specifici per le applicazioni per disaccoppiarle all'interno dello stesso account AWS.**

Un altro importante livello di disaccoppiamento che si può ottenere utilizzando i bus di eventi EventBridge è il disaccoppiamento degli account della AWS Organization. Solitamente le organizzazioni dispongono di più account AWS raggruppati per applicazioni specifiche. In tali contesti, c'è solitamente la necessità di centralizzare risorse specifiche in modo che il team che dovrà lavorarci possa farlo facilmente.

Riprendendo un esempio precedente, si pensi alla necessità di centralizzare i log e monitorare le metriche. Gli event bus di EventBridge possono essere letti/scritti cross-account AWS, per cui applicazioni specifiche potranno facilmente inviare le loro metriche all'event bus dell'account AWS dedicato, dove saranno presenti trigger di eventi per archiviare e consolidare questo tipo di informazioni.

Il servizio Amazon EventBridge dispone di funzioni aggiuntive come **Rules e Pipes** per definire regole per eventi specifici - anche da integrazioni di servizi AWS - filtrarli e trasformarli, per poi consegnarli ai consumatori.

Oltre agli event bus EventBridge, esiste un altro servizio AWS che può essere utilizzato come bus di eventi: Amazon SNS.

Amazon SNS può essere visto come un bus di eventi altamente scalabile e flessibile che consente ai componenti di pubblicare e sottoscriversi agli eventi, assicurando che i cambiamenti in una parte del sistema inneschino risposte appropriate altrove. Utilizzando i topic SNS, possiamo disaccoppiare gli eventi, semplificandone la gestione.

## Code: disaccoppiamento, flusso di controllo e controllo di flusso

Oltre agli event bus, un altro strumento che possiamo usare per raggiungere o migliorare il livello di disaccoppiamento della nostra architettura sono le code. Le code fungono da intermediari tra i componenti, aiutandoli in diversi modi: dalla comunicazione asincrona, che elimina la necessità di avere entrambi i sistemi online nello stesso momento, al disaccoppiamento temporale, che consente ai produttori e ai consumatori di creare ed elaborare i messaggi secondo il proprio ritmo.

Il servizio di code in AWS è Amazon SQS, che può anche essere integrato con EventBridge, complementando il servizio e agendo anche come traduttore da architetture event-driven a message-driven.

Per approfondire il potenziale del servizio EventBridge e il modo in cui possiamo sfruttarlo per il disaccoppiamento architetturale, dobbiamo introdurre due concetti aggiuntivi: **il flusso di controllo e il controllo del flusso**.

Il **flusso di controllo** definisce l'ordine delle operazioni per elaborare i messaggi o i task. Questo argomento è fondamentale per comprendere meglio ogni integrazione con gli event bus in generale. Lo vedremo applicato ai casi d'uso con EventBridge.

Fondamentalmente, dobbiamo definire come i sistemi si integrano tra loro. Possiamo classificare ogni componente di una data integrazione in 4 classi diverse, a seconda della comunicazione con gli altri componenti dell'integrazione:

- **Puller:** riceve attivamente i messaggi; i suoi output sono presi passivamente dal passo successivo.
- **Pusher:** i suoi input sono spinti dal passo precedente, spinge attivamente i messaggi di uscita.
- **Coda:** riceve passivamente i messaggi in ingresso e fornisce quelli in uscita.
- **Driver:** riceve attivamente i messaggi di input e fornisce quelli di output.

Vediamolo in pratica con due semplici esempi che utilizzano un'integrazione molto comune:

Amazon SNS → Amazon EventBridge → destinazione evento.

Poiché Amazon SNS è una sorgente pusher (spinge attivamente gli input), il filtraggio e le trasformazioni degli eventi di EventBridge sono anch'essi passi pusher (quando ricevono l'input e lo elaborano, inoltrando al passo successivo), diversi tipi di destinazioni possono portare a diversi tipi di integrazioni:

1. Destinazione SQS: è una coda (riceve passivamente gli input). L'integrazione avviene senza problemi.
2. Destinazione API: le API possono essere viste come uno step di tipo driver (prende gli input e manda gli output attivamente) e ciò diventa un problema se lo step precedente è di tipo Pusher.

Per ottenere la seconda integrazione, è necessario comprendere il flusso di controllo e applicare strategie adatte. È necessario interporre un componente aggiuntivo per invertire il flusso di controllo, prendendo gli input spinti attivamente dalla sorgente e preparando gli output per una destinazione attiva di tipo poller.

Questa è la definizione di uno step di tipo coda. Una delle proprietà delle code nel flusso di controllo è quella di invertire il flusso, consentendo così integrazioni tra sistemi pusher-puller.

Inoltre, le API possono avere dei limiti (richieste/secondo, ecc...), quindi, impostazioni come la frequenza di invocazione e il TTL dei messaggi in coda possono essere utilizzate per non sovraccaricare il sistema di destinazione.

È qui che entra in gioco il **controllo del flusso**.

Le code sono ottime per la loro proprietà di disaccoppiamento temporale; la velocità dei messaggi prodotti può variare molto da quella dei messaggi consumati. Tuttavia, se queste sono molto diverse, può diventare un problema, poiché la coda si riempirà continuamente fino ad essere completamente piena.

Per risolvere questo problema, abbiamo due principali pattern del controllo del flusso: TTL (time to live), che scarta i messaggi troppo vecchi, e back pressure, che rallenta la velocità di arrivo. Il primo è già implementato in Amazon SQS, mentre il secondo metodo deve essere implementato dal lato produttore.

## Semantica dell'ordine e della consegna

Parlando di sistemi distribuiti e di come i loro componenti si scambiano messaggi, vale la pena spendere qualche riga per discutere l'ordine dei messaggi e la semantica di consegna.

Il mantenimento dell'ordine di elaborazione dei messaggi nei sistemi distribuiti può essere essenziale per garantire la coerenza e l'integrità dei processi aziendali. L'ordinamento dei messaggi consiste nel garantire che gli eventi vengano consumati ed elaborati nello stesso ordine in cui sono stati prodotti.

**L'ordine** diventa una sfida difficile quando iniziamo ad avere più consumatori. Possiamo ottenere l'ordine dei messaggi utilizzando il pattern First-In-First-Out (FIFO) (in poche parole: “chi prima arriva meglio alloggia”.

Gli event bus di EventBridge consegnano gli eventi garantendo un ordine FIFO. Amazon SNS ha topic FIFO, mentre Amazon SQS ha code FIFO e gruppi di messaggi. I gruppi

consentono di ottenere un ordinamento locale dei messaggi, relativamente al gruppo. Infatti, ogni messaggio all'interno di un dato gruppo, secondo l'ordine di arrivo, sarà inviato per l'elaborazione allo stesso consumatore.

Una nota aggiuntiva sui topic e le code FIFO: i gruppi di messaggi sono mantenuti tra topic e coda.

Parlando di **semantica di consegna**, EventBridge supporta due semantiche: "Almeno una volta" e "Esattamente una volta".

La prima scelta fa pensare alla possibilità di eventi duplicati, pertanto è necessario applicare strategie di deduplicazione e progettare i consumatori in modo che siano idempotenti. È possibile ottenere la deduplicazione con le code SQS utilizzando due funzionalità: l'ID di deduplicazione dei messaggi e il timeout di visibilità.

## Gestione degli errori, archivi e repliche

I guasti possono verificarsi anche nei sistemi distribuiti a causa della loro natura complessa.

Una gestione efficace degli errori è essenziale per preservare l'affidabilità del sistema, indipendentemente dalla causa dell'errore (problemi di rete, servizi non disponibili o anomalie impreviste dei dati).

Gli errori possono essere classificati in due categorie principali: gli errori temporanei, che si risolvono con il tempo, come ad esempio l'indisponibilità di un sistema o il carico troppo elevato in quel dato momento, e gli errori sistematici, dovuti a bug che si ripetono continuamente.

Possiamo usare le Dead Letter Queues (DLQ), insieme al parametro MaxRetryAttempt, per distinguere i due tipi di errore e, nel caso di un errore sistematico, far sì che un team lo ispezioni e aggiorni la logica del codice per risolverlo.

I messaggi possono essere archiviati per la verifica, la conformità e l'analisi storica. È possibile archiviare gli eventi a lungo termine utilizzando Amazon S3 per la memorizzazione e utilizzare le DLQ per gli errori immediati. Una volta archiviati gli eventi, è possibile riprodurre i messaggi necessari.

La capacità di riprodurre gli eventi fornisce un ciclo di miglioramento continuo, consentendo agli sviluppatori di imparare dagli errori, perfezionare la logica del sistema e migliorarne l'affidabilità complessiva.

Un consiglio veloce: attenzione a non usare gli ID dei messaggi AWS, poiché cambiano durante la rielaborazione. Inoltre, i consumatori, insieme ai sistemi a valle, rielaboreranno lo stesso evento. Pertanto, devono essere progettati per essere idempotenti.

## Conclusioni

Al termine di questa esplorazione sulle complessità delle architetture cloud disaccoppiate su AWS, arriviamo a un punto di convergenza tra resilienza e innovazione.

I servizi AWS come Amazon EventBridge e Amazon SQS ci aiutano a orchestrare e disaccoppiare la nostra infrastruttura, aprendo la strada a una nuova era del cloud computing in cui affidabilità, scalabilità e flessibilità operano armonicamente insieme.

Nel corso della nostra indagine, abbiamo visto come il disaccoppiamento liberi i componenti in modo che possano operare insieme in modo armonioso, ma *separato*.

Abbiamo analizzato le strategie di flusso di controllo e controllo di flusso e ci siamo resi conto di quanto siano importanti per bilanciare la messaggistica e la comunicazione tra i componenti nei sistemi distribuiti.

Abbiamo anche esplorato rapidamente argomenti come l'ordinamento dei messaggi e la semantica di consegna, concludendo parlando del potenziale offerto dalle strategie di gestione degli errori, dell'uso delle DLQ come reti di sicurezza per l'infrastruttura e delle repliche degli eventi come memoria storica per il miglioramento continuo.

Diventa quindi chiaro come il disaccoppiamento non sia solo una strategia tecnica, ma uno strumento che consente alle organizzazioni di abbracciare l'innovazione, di adattarsi rapidamente a condizioni mutevoli e di creare sistemi duraturi, resilienti ed efficienti.

Terminando questa disamina, continuiamo a innovare, ad adattarci e a liberare l'infinito potenziale delle architetture cloud disaccoppiate su AWS!

### Risorse

- AWS re:Invent 2023 - Advanced integration patterns & trade-offs for loosely coupled systems (API309)

---

## 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!

---

**Matteo Goretti**

DevOps Engineer @ beSharp. Appassionato di Cloud Computing e Intelligenza Artificiale, in particolare, Machine Learning e Deep Learning. Amo il trekking e la natura in generale. Mi rilasso con la mia chitarra, giocando ai videogames o guardando serie TV.

---