3 modi per disaccoppiare i tuoi microservizi: code SQS, load balancing ELB e sistema di notifiche SNS

12 Maggio 2023 - 10 min. read

Amazon Simple Notification Service (SNS) | Amazon Simple Queue Service (SQS) | Application Modernization | AWS Elastic Load Balancer (ELB) | Microservices

Perché è così importante disaccoppiare i servizi? E perché dovremmo evitare architetture tightly coupled?

Le care vecchie architetture monolitiche ci hanno servito bene e hanno avuto i loro momenti di gloria. Questo perché nei primi periodi di vita di internet le aziende necessitavano di un unico grande sistema che permettesse di gestire e distribuire il contenuto del loro sito o della loro applicazione. Un'architettura monolitica, però, fornisce una soluzione all-in-one, fortemente "accoppiata", ovvero con un'unica architettura in grado di gestire e pubblicare tutto il contenuto sul web.

In questo modo tutti i processi funzionano come un singolo servizio.

Aggiungere o migliorare le funzionalità di un'applicazione monolitica diventa più complesso man mano che questa cresce di dimensione. Utilizzarle, inoltre, comporta un rischio dal punto di vista della disponibilità costante dell'applicazione poiché, essendo formata da molteplici processi dipendenti e strettamente accoppiati, aumenta l'impatto che un guasto di un singolo processo può avere.

Quindi, ecco il problema: se il grande cervello monolitico si guasta, si guasta tutto.

Per risolvere questo problema possiamo trasformare la cara vecchia architettura monolitica in un'architettura moderna costituita da **microservizi**. Questi non sono altro che un approccio architettonico e organizzativo allo sviluppo del software che diventa composto

da piccoli servizi indipendenti, i quali comunicano attraverso API ben definite e svolgono una singola funzione.

Essendo gestiti in modo indipendente, ogni servizio può essere aggiornato, distribuito e scalato per soddisfare la domanda di specifiche necessità di un applicativo, il tutto senza influenzare il funzionamento degli altri servizi. Inoltre, lavorando su un singolo servizio, più sviluppatori possono contribuire allo sviluppo del codice e, se questo diventa troppo complesso nel tempo, può essere suddiviso, a sua volta, in servizi più piccoli.

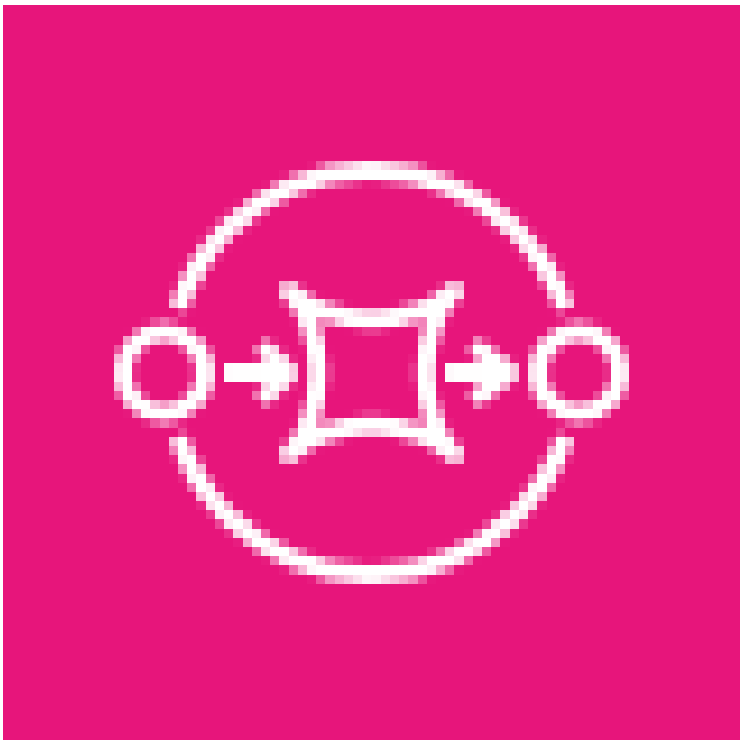
In questo articolo spiegheremo un caso d'uso ispirato a uno scenario reale. Dimostreremo come risolvere alcuni problemi comuni utilizzando tre servizi AWS con l'obiettivo di disaccoppiare la nostra architettura.

Li utilizzeremo in diverse combinazioni per superare diversi problemi.

- SQS (Simple Queue Service)
- SNS (Simple Notification Service)
- ELB (Elastic Load Balancer)

Prima di procedere con la spiegazione facciamo un breve riassunto dei microservizi e di come funzionano:

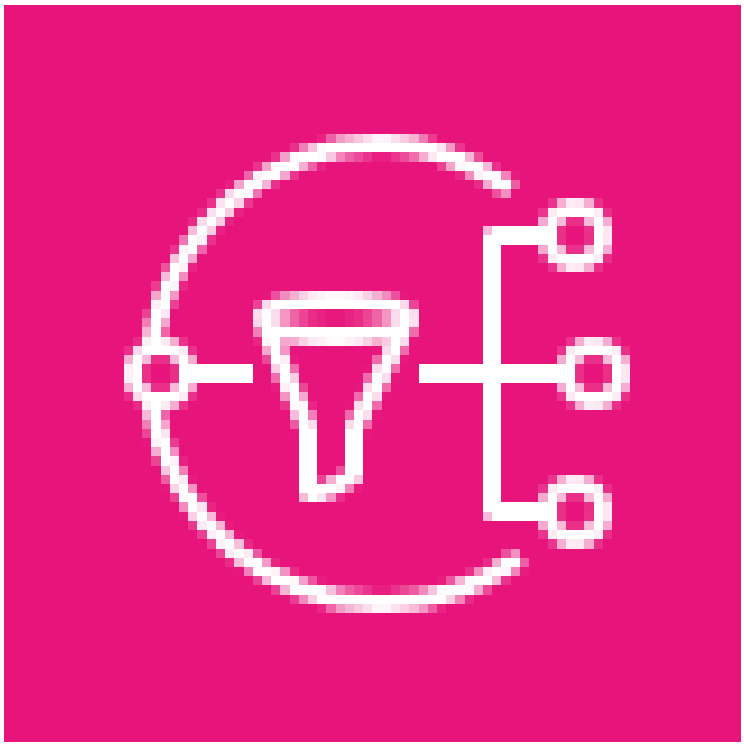
SQS (Simple Queue Service)



Ehi! Non è la prima volta che parliamo di come utilizzare SQS per disaccoppiare i servizi, l'abbiamo già fatto in **questo articolo**!

In breve, SQS è un servizio che mette a disposizione delle code di messaggi. Può inviare, memorizzare e ricevere messaggi tra componenti software. Ciò consente di disaccoppiare i servizi in modo affidabile e di inviare e ricevere elevati volumi di dati senza che altri servizi siano attivi per farlo.

SNS (Simple Notification Service)

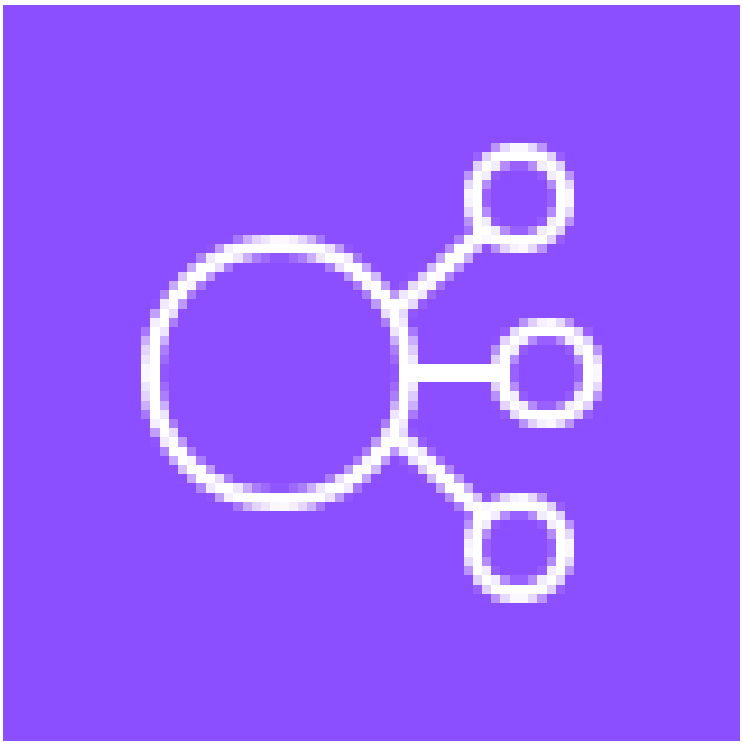


SNS è un servizio gestito che fornisce la consegna dei messaggi dal publisher al subscriber. L'editor invia il messaggio ad un “topic”. Il client può quindi iscriversi al topic SNS e ricevere una notifica ogni volta che un messaggio viene inviato su di esso.

Il messaggio può essere ricevuto in modi diversi, a seconda del tipo di endpoint utilizzato.

Può essere ricevuto da altri servizi AWS come Lambda Functions, SQS, Kinesis, HTTP o anche come e-mail, notifiche push e SMS.

ELB (Elastic Load Balancer)



Un Elastic Load Balancer permette di distribuire automaticamente il traffico in entrata a più tipi di target. Un utilizzo comune è quello di far scalare automaticamente il numero di istanze EC2 che elaborano il traffico in entrata al fine di ottenere una maggiore disponibilità e tolleranza agli errori.

Esistono tre tipi di Elastic Load Balancer (ELB) su AWS:

1. Application Load Balancer (ALB): è un ELB progettato per il traffico HTTP/HTTPS al livello applicazione del modello OSI. Instrada le richieste in base al loro contenuto e si adatta meglio alle architetture basate su microservizi e container.
2. Network Load Balancer (NLB): gli NLB sono progettati per il traffico TCP/UDP e operano al livello di trasporto del modello OSI. Sono in grado di gestire milioni di richieste al secondo con una bassa latenza.
3. Il Classic Load Balancer (CLB): è il bilanciatore di carico originale di AWS che opera sia a livello di applicazione che di trasporto del modello OSI. Può gestire il traffico HTTP/HTTPS, TCP e SSL, il che lo rende ideale per le architetture semplici che necessitano di soluzioni di bilanciamento del carico di base. Attualmente è deprecato, quindi è necessario utilizzare uno degli altri due tipi.

Differenze tra disaccoppiamento asincrono e sincrono

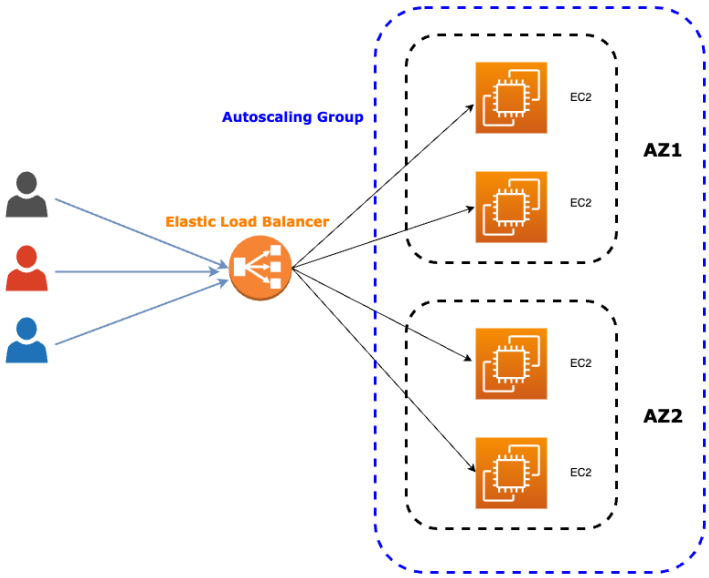
Il disaccoppiamento dei microservizi può essere fatto in modo sincrono o asincrono.

Il disaccoppiamento sincrono richiede che tutti i processi siano strettamente disaccoppiati ed eseguiti come un singolo servizio. Quando si riceve una richiesta, tutti i processi rispondono in modo sincrono.

Il disaccoppiamento asincrono, invece, separa i processi e consente loro di operare indipendentemente l'uno dall'altro.

Se determinate operazioni possono essere eseguite in modo asincrono e non necessitano di essere completate in una singola transazione, una parte dell'architettura può raccogliere le richieste, mentre un'altra parte può elaborarle successivamente o quando le risorse diventano disponibili.

Ora illustreremo uno scenario reale in cui andremo a modificare un'architettura disaccoppiandone i componenti al fine di migliorare le prestazioni, l'affidabilità e l'esperienza dell'utente. Per fare questo utilizzeremo i servizi AWS precedentemente menzionati.



Il problema

Immaginiamo di avere un piccolo sito social network che sta diventando popolare tra i giovani. Il sito consente agli utenti di caricare immagini, le quali devono essere ridimensionate in diversi formati per garantire un utilizzo ottimale su tutti i tipi di dispositivi. Tuttavia, non vogliamo limitarci a ridimensionare le immagini. Vogliamo anche applicarci un modello di machine learning per estrarne alcune caratteristiche in modo da taggarle con informazioni legate al contenuto dell'immagine. In questo modo gli utenti potranno cercare le immagini in base al contenuto, piuttosto che utilizzare il nome dei file.

Il numero di immagini caricate in un mese può variare notevolmente, quindi abbiamo bisogno di una soluzione che sia efficiente in termini di costi e affidabile durante i picchi di richieste. La nostra soluzione di partenza è costituita da una singola istanza EC2 che gestisce sia la logica di backend del social network, sia il processo di ridimensionamento delle immagini, sia l'esecuzione del modello di apprendimento automatico per il tagging delle foto.

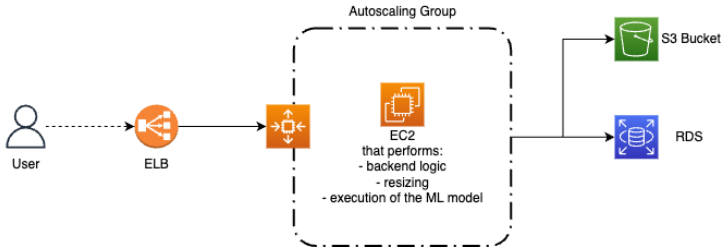
Questa soluzione è stata sufficiente e ha funzionato bene fino a che il sito ha avuto pochi utenti, ora non lo è più.

Con l'aumento del numero di iscritti, il carico sull' istanza EC2 è aumentato. È chiaro che esporre una singola istanza EC2 a tutto il traffico stia diventando insostenibile. Le chiamate API e le elaborazioni delle immagini diventano sempre più lente, rendendo gli utenti frustrati e scontenti, quasi tentati di abbandonare il nostro fantastico social network. Per risolvere questi problemi, dobbiamo far scalare la nostra infrastruttura in modo appropriato. Consideriamo quindi di utilizzare un Elastic Load Balancer per distribuire il traffico su più istanze EC2. Questo aiuterebbe a garantire che non ci siano colli di bottiglia nel sistema e che gli utenti possano caricare e accedere alle immagini in modo rapido e semplice.

Consideriamo anche di utilizzare un Amazon S3 Bucket per archiviare le immagini. Questo ci permette di sfruttare l'archiviazione economica di Amazon e di garantire che il sito web rimanga reattivo anche durante i picchi di richieste di caricamento delle immagini.

Adottando queste misure, possiamo garantire che il nostro sito web rimanga veloce e reattivo, anche quando la nostra base di utenti continua a crescere.

Prima soluzione - ELB



Come detto in precedenza, per poter gestire un numero crescente di utenti, dobbiamo scalare la nostra architettura per ottenere prestazioni migliori. La nostra soluzione consiste nell'utilizzare un Elastic Load Balancer con un AutoScaling group per aumentare il numero di istanze EC2 che elaborano le immagini.

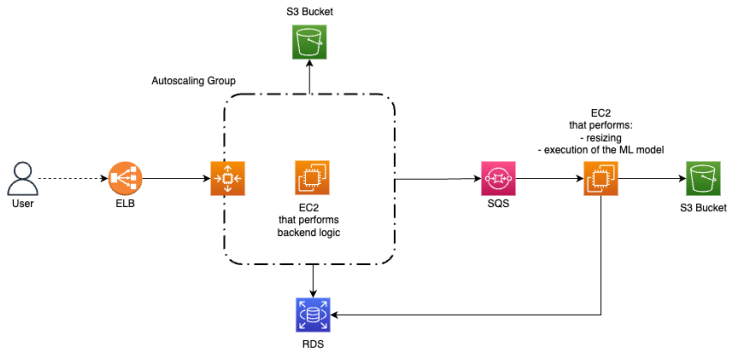
L'Elastic Load Balancer riceve tutto il traffico in entrata e lo reindirizza all'AutoScaling group. Così facendo possiamo ricevere tutto il traffico con l'ELB invece che con l'EC2. Collegandolo all'AutoScaling group vengono generate più istanze man mano che il traffico aumenta. Si tratta di una soluzione sincrona poiché il ridimensionamento dell'immagine e l'applicazione del modello di machine learning avvengono quando viene ricevuta la richiesta.

Abbiamo fatto il primo passo verso il disaccoppiamento dei servizi della nostra soluzione. Tuttavia, le istanze EC2 generate rispondono in modo sincrono quando vengono chiamate,

con un conseguente impatto negativo sull'esperienza dei nostri utenti. L'attesa per l'elaborazione dell'immagine può essere lunga, e noi vogliamo che i nostri utenti siano felici.

Fortunatamente, al momento in cui l'immagine viene caricata, il ridimensionamento dell'immagine e l'inserimento dei tag non sono necessari. Introducendo del disaccoppiamento asincrono, utilizzando le code, potremo ulteriormente disaccoppiare il nostro sistema. Possiamo ottenere questo risultato con il secondo dei nostri servizi, SQS.

Seconda soluzione - SQS



Prima di tutto, dovremmo separare la logica del sito web e del backend dalla logica dell'algoritmo di apprendimento automatico e dalla logica di ridimensionamento delle immagini, per migliorare l'efficienza e snellire i processi. Questo ci permetterà di gestire meglio le diverse richieste e di garantire che ogni componente funzioni in modo ottimale.

Grazie a questa separazione otterremo due tipi di istanze EC2 che svolgeranno compiti distinti. La prima sarà responsabile di rispondere direttamente alle richieste degli utenti, mentre la seconda gestirà tutte le attività di processing delle immagini. Dividendo il carico di lavoro in questo modo, possiamo garantire che ogni istanza EC2 sia ottimizzata per il suo compito specifico.

La prima istanza EC2 sarà collocata dietro un ELB con AutoScaling. Quando vengono create nuove istanze EC2, queste raccolgono le immagini e le salvano su S3, e salvano su RDS i loro metadati (come l'ID, la data e l'ora di caricamento, le dimensioni e così via), e nel frattempo scrive sulla coda quali immagini deve elaborare la seconda istanza.

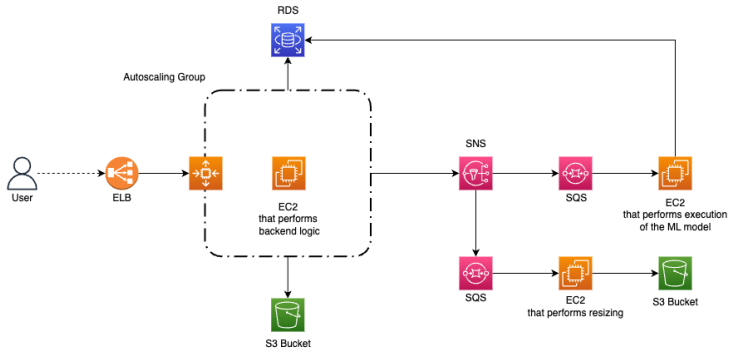
La seconda istanza EC2 controllerà se nella coda sono presenti dei messaggi da processare per poter effettuare il resizing delle immagini. Queste saranno memorizzate in un Bucket S3, il quale potrebbe anche essere lo stesso che abbia già utilizzato ma con path diversi.

La seconda istanza EC2 eseguirà anche l'algoritmo di tagging, analizzando l'immagine e registrando i tag associati nell'RDS come attributi aggiuntivi dell'immagine originale, insieme agli altri metadati dell'immagine.

Bonus tips

Poiché abbiamo disaccoppiato il nostro sistema, e non abbiamo più bisogno che il secondo tipo di istanza EC2 sia sempre disponibile, possiamo utilizzare le Spot Instances per risparmiare fino al 90% sui costi totali. L'immagine verrà caricata immediatamente dall'istanza EC2 che gestisce la logica di backend. L'elaborazione dell'immagine, sia il ridimensionamento che il tagging, può avvenire così in modo asincrono grazie alla nostra architettura disaccoppiata.

Terza soluzione - SNS + SQS



Siamo riusciti a disaccoppiare la logica di backend dall'elaborazione delle immagini, questa ora può essere eseguita in modo asincrono. Ciò significa che adesso la nostra architettura può gestire più richieste di elaborazione delle immagini contemporaneamente, senza ritardare l'esecuzione della logica di backend. Tuttavia, cosa succede se il processo di ridimensionamento è molto più veloce dell'esecuzione del modello di machine learning? Per ottimizzare ulteriormente il processo e ottenere un'architettura a microservizi altamente disaccoppiata, dobbiamo dividere ulteriormente la logica di elaborazione in altri due nuovi sistemi indipendenti.

Il primo sistema è responsabile del processo di ridimensionamento, mentre il secondo esegue il modello di machine learning. Questo assicura che i due processi possano essere eseguiti in modo indipendente e asincrono, senza influenzarsi a vicenda.

Per informare entrambe le parti quando viene caricata una nuova immagine e avviare contemporaneamente le due elaborazioni, possiamo utilizzare Amazon SNS. L'istanza EC2 che gestisce la logica di backend scrive su un topic SNS, entrambe le parti di elaborazione sono subscriber del topic e ricevono una notifica per capire quando deve essere avviata l'esecuzione della logica di elaborazione.

Inoltre, possiamo migliorare l'efficienza del sistema introducendo un SQS davanti a entrambe le unità EC2 di elaborazione, questo permette di disaccoppiare ulteriormente le

singole parti della nuova architettura per gestire nel modo giusto i picchi di caricamento delle immagini.

Per concludere, dividendo la logica di elaborazione in due parti indipendenti, e introducendo l'Elastic Load Balancer, possiamo ottenere un'architettura a microservizi altamente disaccoppiata in grado di gestire in modo efficiente e rapido più richieste di elaborazione di immagini.

Final tip

In tutti i nostri esempi, abbiamo sempre usato istanze EC2 come unità di elaborazione. Tuttavia, possiamo diventare completamente serverless riscrivendo la logica di backend, la logica di ridimensionamento e la logica di tagging per poterle utilizzare come diverse funzioni Lambda.

Una funzione Lambda può anche essere collocata dietro un ELB, come un Application Load Balancer. Inoltre è in grado di gestire automaticamente lo scaling del numero di occorrenze in esecuzione fino al raggiungimento del limite massimo di concorrenza del proprio account (non è necessario un gruppo di AutoScaling).

Tuttavia spiegare il funzionamento delle funzioni Lambda, e quando utilizzarle in sostituzione di un'istanza EC2, va oltre lo scopo di questo articolo.

Conclusione

In questo articolo abbiamo esplorato tre pattern comuni per disaccoppiare i microservizi della vostra architettura cloud. Utilizzando tre semplici servizi AWS, è possibile aumentare la scalabilità e le prestazioni della soluzione, eliminando al contempo molti problemi comuni. Il problema più significativo è il mantenimento di un' architettura monolitica. Il disaccoppiamento dei microservizi consente di aggiornare e correggere facilmente parti specifiche dell'architettura senza influire sulle altre. Inoltre, grazie a questa architettura flessibile, è possibile gestire meglio le problematiche legate alle prestazioni e ai picchi di richieste.

Avete mai provato a disaccoppiare i microservizi della vostra architettura? Se sì, ci piacerebbe conoscere la vostra esperienza. Sentitevi liberi di condividerla nei commenti qui sotto! Se no, cosa state aspettando a provarci? Arrivederci al prossimo articolo su **Proud2beCloud**!

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!

Giacomo Zagami

MLOps/DevOps @ beSharp. Al lavoro sono un ingegnere biomedico alto e magro con la passione per la matematica, i dati ed il machine learning, ma in realtà sono un chitarrista, amo la musica e suonare con altri musicisti.

Antonio Minolfi

Cloud Native Developer @ beSharp. Curioso di natura, amo imparare e scoprire qualcosa di nuovo ogni giorno. Mi piace sviluppare soluzioni che semplificano compiti complessi e utilizzare le tecnologie più recenti per farlo. Nel tempo libero mi rilasso andando in palestra o giocando ai videogiochi ("Gains and Games!")
