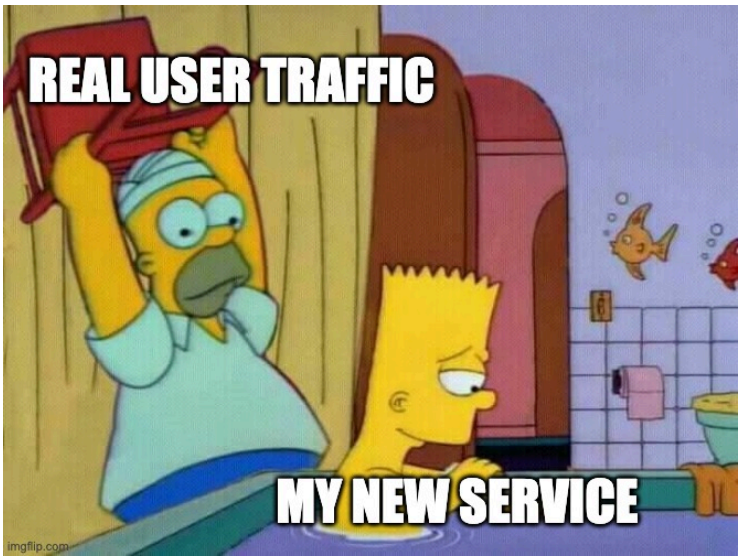
Testing del Codice con Traffico Reale: Oltre lo Staging con AWS VPC Traffic Mirroring

14 Luglio 2025 - 6 min. read

Ci siamo passati tutti: gli unit test sono tutti “verdi”, gli integration test passano senza problemi e perfino i test di carico più rigorosi confermano che il nuovo codice è pronto. La fiducia è alle stelle. Eppure, una domanda assillante continua ad aleggiare nell'aria:

"cosa succederà con gli utenti *reali*? Con picchi di traffico imprevedibili che ti fanno esclamare: “Non sapevo nemmeno che l'API potesse essere usata così!”



La questione ci è particolarmente cara. Avevamo appena completato un importante refactoring di un'applicazione critica, di quelle il cui minimo intoppo provoca ripercussioni sul business.

Lo Scenario: L’architettura in Produzione

La nostra configurazione segue un pattern comune in AWS: un AWS Application Load Balancer distribuisce il traffico su una flotta di istanze Amazon EC2 che ospitano il

nostro servizio. La funzione principale del servizio? Interrogare un database MongoDB e restituire i risultati. Semplice, ma fondamentale.

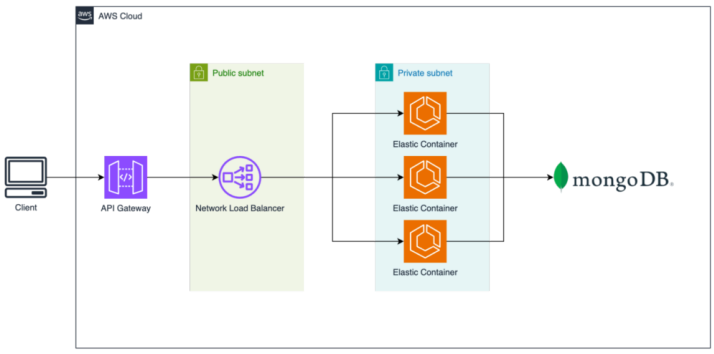


Diagramma semplificato del flusso di produzione

"What If": Replicare la Produzione Senza Rischi

Per dissipare ogni dubbio, abbiamo indetto una sessione di brainstorming (alimentata, come da tradizione, da una generosa dose di caffè). "E se" propone qualcuno, "potessimo *spiare* in modo trasparente il traffico di produzione e inviarlo al nostro nuovo codice, in un ambiente completamente isolato da quello reale?".

Sembra un'utopia, ma AWS offre un servizio adattabile al nostro scopo: **Amazon VPC Traffic Mirroring**.

Immaginate uno specchio unidirezionale. Da un lato, il traffico di produzione scorre normalmente, gestendo le richieste degli utenti senza interruzioni. Dall'altro, una copia esatta di quel traffico viene inviata a una destinazione designata, come un Network Load Balancer o un'interfaccia di rete.

Il dettaglio cruciale è *come* viene inviata questa copia. I pacchetti replicati sono incapsulati in VXLAN, un protocollo di virtualizzazione di rete che wrappa frame Ethernet di livello 2 in pacchetti UDP per il trasporto su reti IP. Questo impedisce loro di essere trattati come traffico di rete standard e di interferire con qualsiasi altro sistema. Tuttavia, significa anche che la nostra applicazione di test non può processare questi pacchetti VXLAN come normali richieste HTTP.

Per colmare questa lacuna, abbiamo bisogno di una semplice utility per:

1. **Ricevere** i pacchetti incapsulati in VXLAN.

2. **Decapsularli** per estrarre le richieste di produzione originali.
3. **Inoltrare** le richieste al nostro ambiente di test.

Con questo approccio, il flusso di produzione resta completamente isolato e ignaro dell'ambiente parallelo, mentre il nostro ambiente di test riceve una replica in tempo reale del traffico senza alcun rischio. Non è una soluzione pronta all'uso, ma con lo strumento giusto per la decapsulazione, si è rivelata un modo potente e sicuro per validare il nostro nuovo codice contro la realtà.

Costruire il nostro Ambiente "Doppelgänger"

Con una strategia chiara, ci siamo messi al lavoro.

All'inizio abbiamo costruito una replica isolata. Abbiamo rilasciato uno stack completo e parallelo: nuove istanze Amazon EC2 con il codice refattorizzato, un AWS Application Load Balancer "ombra" dedicato e una nuova istanza MongoDB popolata con uno snapshot recente della produzione. Questo era il nostro campo di prova ad alta fedeltà.

Poi abbiamo deployato il nostro ambiente di mirroring. Abbiamo configurato un AWS Network Load Balancer davanti a un AWS Auto Scaling group di istanze Amazon EC2. Su queste istanze, abbiamo installato uno strumento open-source (come questo) per gestire la decapsulazione VXLAN e inoltrare il traffico al nostro AWS ALB ombra. Questo gateway era il ponte tra il mondo mirrorato e il nostro ambiente di test.

Non avevamo bisogno di mirrorare *tutto* il traffico; le comunicazioni operative avrebbero solo aggiunto rumore e costi. Il nostro servizio opera sulla porta 8080, quindi abbiamo creato un semplice **Filtro di Mirroring** (Mirror Filter): "Duplica solo il traffico TCP destinato alla porta 8080". Questo ha focalizzato il nostro esperimento e, soprattutto, ci ha aiutato a gestire i costi.

Infine, abbiamo creato una **Sessione di Mirroring** (Mirror Session) per collegare la sorgente, il filtro e la nostra destinazione (il gateway). E abbiamo premuto l'interruttore.

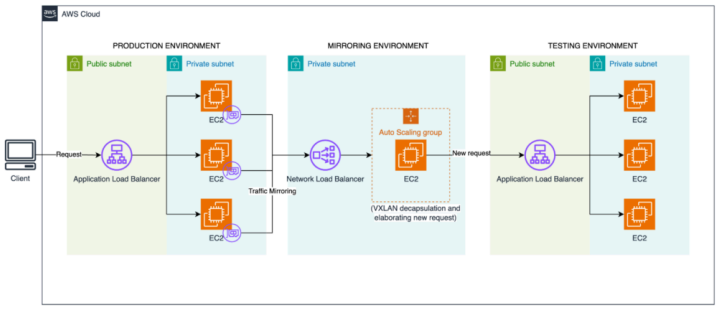


Diagramma dell'architettura finale con il traffic mirroring

Il Momento della Verità

I risultati sono stati immediati e illuminanti. Osservando le nostre dashboard di CloudWatch, abbiamo visto le metriche di CPU e Memoria del nuovo servizio di test iniziare a fluttuare, imitando perfettamente il ritmo dell'ambiente di produzione. La nostra applicazione ombra era **viva** e stava processando una replica in tempo reale del carico di produzione.

È qui che è emerso il vero valore.

Abbiamo lasciato il mirroring attivo per un'intera giornata, catturando un ciclo completo di traffico, dai picchi alle ore di calma. L'analisi dei log dell'applicazione e delle metriche del database ha portato alla luce due aspetti:

- **Un problema di query nascosto**: Il nostro codice refattorizzato aveva inavvertitamente introdotto un pattern di query N+1. Era abbastanza performante da passare inosservato nei test sintetici, ma sotto il carico eterogeneo degli utenti reali, stava generando migliaia di piccole e inutili chiamate al database. Sarebbe stato quasi impossibile da individuare in un ambiente di staging tradizionale.
- **Conferma delle prestazioni**: Dopo aver corretto il bug e ridistribuito il servizio ombra, le nuove metriche hanno fornito una prova innegabile: il nostro refactor era,

di fatto, significativamente più efficiente. L'utilizzo della CPU era inferiore e il carico sul database era notevolmente diminuito.

Avevamo superato il confine delle supposizioni. Avevamo dati concreti, validati sul campo, senza aver mai messo a rischio un singolo utente. Il successivo deployment in produzione è andato liscio come l'olio.

Key Takeaways dal Traffic Mirroring

Questa tecnica è diventata parte integrante della nostra strategia per i rilasci critici. Non sostituisce i test standard, ma funge da ultimo controllo pre-lancio.

Ecco cosa ci siamo portati a casa da questa esperienza:

- **La gestione dei costi è cruciale**: Il traffic mirroring ha un costo per GB processato. Siate *frugali*, usate filtri aggressivi per isolare solo il traffico che vi serve e limitate gli esperimenti a finestre temporali definite per ottimizzare la spesa.
- **La sicurezza non è negoziabile**: il traffico replicato è traffico di produzione. Nel nostro caso, il TLS veniva terminato a livello del Load Balancer di produzione, quindi il traffico specchiato non era cifrato. Se da un lato questo facilita l'analisi, dall'altro impone che l'ambiente ombra sia sicuro, privato e con controlli di accesso rigidi. Trattate i dati replicati con la stessa cura dei dati di produzione.
- **Un alleato, non un sostituto**: Questa è una tecnica di validazione finale, non un sostituto dei test iniziali. Unit test, integration test e ambienti di staging rimangono pilastri fondamentali. Il traffic mirroring è la ciliegina sulla torta di una solida pipeline di rilascio.

Amazon VPC Traffic Mirroring ha trasformato il nostro modo di affrontare i deployment. Ha colmato il divario tra le nostre aspettative e la realtà della produzione. Abbiamo smesso di *sperare* che il nostro codice reggesse l'impatto con il mondo reale: ora sappiamo che lo farà.

La prossima volta che affrontate un rilascio critico, chiedetevi: e se poteste vedere come si comporterà il nuovo sistema con il traffico reale, *prima* che vada online?

Siamo curiosi di conoscere le vostre esperienze. Qual è il vostro approccio per promuovere il codice in produzione e mitigare i rischi? Condividete le vostre strategie e le vostre esperienze 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!

Alessandro Gallo

DevOps Engineer @ beSharp e appassionato di tecnologia fin da molto prima che diventasse un trend. Nel tempo libero mi troverete immerso in libri fantasy e film. In un passato non troppo lontano sono stato anche uno chef; oggi "preparo" soluzioni in Cloud!
