Anti-pattern FinOps: cosa (non) fare per avere più budget per innovare

22 Ottobre 2025 - 9 min. read

FinOps

Eccoci con un altro portmanteau tra "*qualcosa*" e *Operations.* Questa volta tocca al tema Finance.

Generalmente, il primo impatto con il termine **FinOps** è affrontato con una certa superficialità: si da per scontato che se *Operations* riguarda l'infrastruttura e *Finance* significa "$$", allora l'unione delle due deve per forza voler dire *"ridurre gli sprechi nel cloud"*.

Giusto?
Beh, non proprio. O meglio, non solo.

Ridurre i costi è certamente una parte del gioco, ma fermarsi qui sarebbe come pensare che DevOps significhi solo "automatizzare le build". In realtà, FinOps è molto di più; è un **approccio culturale e operativo che unisce team tecnici, finanziari e di business per massimizzare il valore del Cloud**, non semplicemente per risparmiare qualche dollaro.

Per fugare ogni dubbio è il caso di spendere un paragrafo per spiegare a grandi linee alcuni concetti di FinOps. Se li conosci già sentiti libero di saltarlo :)

Cosa significa FinOps?

Prima di tutto, vale la pena chiarire un punto fondamentale: FinOps non introduce concetti rivoluzionari. È piuttosto un modo strutturato di riportare in primo piano

l'attenzione all'efficienza. Il framework nasce con l'obiettivo di massimizzare il valore di business del cloud, abilitando decisioni rapide e guidate dai dati e creando una responsabilità finanziaria condivisa tra ingegneria, finanza e business. In questo senso, rappresenta un vero e proprio cambio di paradigma: la gestione dei costi non è più un'attività di consuntivo, ma diventa parte integrante dei processi decisionali quotidiani.

Il percorso FinOps si articola in tre fasi iterative che si alimentano a vicenda: **Inform**, **Optimize** e **Operate**.

Nella fase di **Inform**, l'attenzione è rivolta alla raccolta e alla valorizzazione dei dati relativi a costi, utilizzo ed efficienza del cloud. Questi dati vengono trasformati in strumenti di analisi e reportistica capaci di offrire una visibilità accurata sulle dinamiche di spesa. Ne derivano capacità di budgeting e di previsione, ma anche la possibilità di misurare le performance attraverso KPI e benchmark interni o esterni. La natura elastica e on-demand del cloud, insieme ai meccanismi di sconto, rende indispensabile un monitoraggio continuo: solo con informazioni tempestive è possibile anticipare la spesa, prevenire sorprese e mantenere un ritorno sull'investimento coerente con gli obiettivi aziendali.

La fase di **Optimize** si concentra invece sull'efficientamento. È il momento in cui i dati raccolti vengono utilizzati per individuare opportunità concrete di ottimizzazione, ad esempio ridimensionando le risorse sottoutilizzate, adottando architetture più moderne, automatizzando la gestione dei carichi di lavoro e riducendo gli sprechi.

Anche le tariffe diventano un campo di ottimizzazione, grazie a modelli di sconto come Reserved Instances e Savings Plans. Questa fase non si limita a tagliare i costi, ma punta a un dialogo continuo tra i diversi team, così da garantire che le prestazioni del cloud restino sempre allineate agli obiettivi strategici dell'organizzazione.

Infine, nella fase di **Operate** si consolidano i cambiamenti. Qui trovano spazio la definizione di politiche di governance, il monitoraggio della conformità, la responsabilizzazione delle persone e la diffusione di una cultura della responsabilità condivisa. I team tecnici, finanziari e commerciali collaborano in maniera costante, prendendo decisioni iterative e incrementali basate sulle evidenze raccolte nelle fasi precedenti. L'approccio è ciclico: ogni azione porta a nuove informazioni, che alimentano ulteriori ottimizzazioni e rendono i processi più maturi e consapevoli.

Il valore di FinOps sta proprio in questo ciclo continuo. Informare, ottimizzare e operare non sono momenti isolati, ma fasi che si intrecciano e si ripetono, portando l'organizzazione a una gestione del cloud sempre più evoluta e capace di generare valore concreto.

Tre Anti-pattern comuni

Ora che abbiamo chiarito cosa sia FinOps e come funzioni, vediamo dove le cose possono andare storte. In quanto consulenti, non siamo certo nuovi a situazioni disperate. A volte però, gli errori più subdoli non si manifestano subito: restano silenziosi durante le fasi di progettazione e implementazione per poi rivelarsi solo dopo il rilascio in produzione. Ed è proprio lì, quando l'infrastruttura comincia a scalare e i costi aumentano, che ormai è troppo tardi per intervenire e il conto (letteralmente) arriva.

Questi errori sono la conseguenza di progettazioni che hanno considerato solo i requisiti tecnici, ignorando (o trattando superficialmente) l'impatto economico delle scelte architetturali. Un approccio che contrasta con lo "shift-left" dei costi: portare la consapevolezza finanziaria fin dalle prime fasi di progettazione, quando le decisioni possono ancora fare la differenza.

Ma quanto è bello il serverless?

Diciamocelo: il concetto del serverless è affascinante.

L'idea di non dover gestire infrastruttura (né sistemisti!) è il sogno di ogni sviluppatore.

E così, complici la facilità d'uso e la scalabilità automatica, si finisce spesso per scegliere un'architettura serverless anche quando non è la scelta più adatta.

Per poter prendere delle decisioni informate è necessario avere chiari i casi d'uso.

Il serverless è perfetto per:

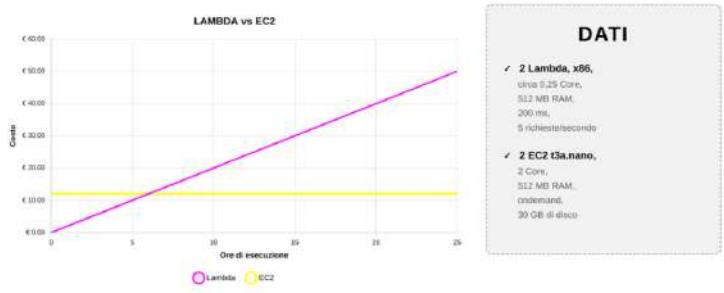
- **Applicazioni con carico variabile**: in queste situazioni la capacità di scaling del serverless danno il massimo.
- **Microservizi o applicazioni event driven**: ogni microservizio può diventare un container gestito o una funzione serverless lanciata da un trigger.
- **Efficienza economica su uso sporadico**: molti servizi serverless possono scalare a zero, rendendo nulli i costi in caso di inutilizzo.

È il caso di farsi qualche domanda in più nel caso in cui si abbiano:

- **Applicazioni con carico costante**: per un utilizzo costante una macchina virtuale costerà sempre meno e performerà anche meglio.
- **Applicazioni che richiedono bassa latenza**: se la latenza è critica, il serverless potrebbe non essere la scelta migliore a causa dei "cold start".
- **Paura del vendor lock-in**: l'approccio serverless porta ad integrarsi maggiormente con il cloud provider, cosa che non sempre i requisiti permettono.

Prendiamo come esempio una versione opportunamente semplificata del confronto tra i costi running di una Lambda Function e di una macchina EC2. Come si può immaginare i costi della Lambda crescono più o meno linearmente con il tempo di utilizzo, mentre i costi della macchina EC2 di un certo taglio sempre accesa sono fissi.

Inoltre è importante considerare che l'EC2 in esempio ha una maggiore potenza computazionale e un disco.



Chiaramente questo ragionamento salta se il carico di lavoro varia molto nel tempo. Una macchina EC2 sempre accesa ha dei costi infinitamente maggiori di una Lambda eseguita sporadicamente.

Più in generale, il *Modeling* è un principio cardine del FinOps che impone di modellare il profilo di carico e l'andamento dei costi nel tempo prima di prendere decisioni architetturali. Questo approccio integra l'analisi economica come un requisito non funzionale (NFR) primario, garantendo che la progettazione iniziale sia sostenibile e allineata all'obiettivo di massimizzare il valore del business a lungo termine.

Come ci ricorda anche Werner Vogels nel suo progetto The Frugal Architect,

"un'architettura ben progettata è quella che bilancia performance, resilienza e costo nel tempo".

In altre parole, essere "frugali" non significa spendere meno, ma spendere meglio costruendo sistemi che scalano in modo sostenibile senza compromessi sulla qualità.

Logging like no tomorrow

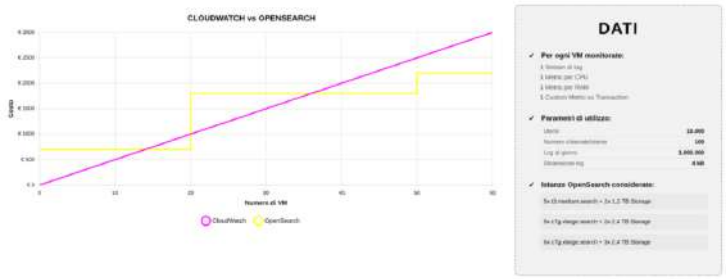
Chi fa il nostro mestiere sa che log e metriche sono fondamentali per comprendere lo stato di salute di un'infrastruttura. Due delle soluzioni più diffuse su AWS per la gestione centralizzata dei log sono **Amazon CloudWatch Logs** e **Amazon OpenSearch Service**. Entrambe offrono funzionalità robuste, ma la scelta tra l'una e l'altra dipende in larga misura dai requisiti specifici del caso d'uso, dal volume dei dati, dalla complessità delle query e dal budget a disposizione.

Nativo, ben integrato e semplice da usare, quasi sempre Cloudwatch è la scelta di default. In fase di kickoff di un progetto, il solo sentir parlare dei costi e della complessità di OpenSearch poi toglie qualunque dubbio. Purtroppo però anche questa volta la soluzione più semplice non è necessariamente la migliore, come sempre la soluzione migliore è: *dipende*.

La differenza cruciale risiede nel modello economico. L'approccio a consumo di CloudWatch può far crescere rapidamente i costi con l'aumento del numero di metriche ad alta cardinalità, della frequenza di campionamento e dei log ingeriti e conservati. Un cluster OpenSearch gestito, al contrario, introduce un costo più "a gradini": si dimensionano i nodi e si assorbe il volume, con un costo che scala in modo più prevedibile. Superata una certa soglia di volume, spostare log ad alto volume e

metriche personalizzate su OpenSearch (mantenendo in CloudWatch l'heartbeat e gli allarmi essenziali) risulta significativamente più efficiente.

Nel seguente esempio è stata considerata un'applicazione di dimensione importante, basata su EC2 in autoscaling e con una media di diecimila utenti giornalieri che effettuano una media di un centinaio di chiamate a testa. Il singolo log record è di dimensione abbastanza elevata: 4kB. Infine è importante considerare che nell'analisi sono state considerate istanze OpenSearch sovradimensionate, in modo da non avere problemi di risorse scarse e permettere operazioni di analisi dei dati.



Dal grafico si nota che per un numero limitato di macchine virtuali Cloudwatch rimane economicamente vantaggioso, ma quando le istanze sono in numero medio/elevato il costo di OpenSearch raggiunge un plateau, mentre quello di Cloudwatch continua a salire linearmente.

Un requisito non funzionale specifica i criteri che possono essere utilizzati per giudicare il funzionamento di un sistema (accessibilità, disponibilità, scalabilità, ...), ma ciò che spesso viene trascurato è il costo. I progetti possono fallire perché non si considerano i costi in ogni fase del business: dalla progettazione allo sviluppo fino all'operazione.

In ottica *Frugal Architect*, questo significa introdurre la consapevolezza economica già nella fase di design: ogni metrica raccolta, ogni retention policy e ogni log stream deve essere intenzionale. "**Measure everything but pay attention to what you keep**", direbbe Vogels.

Tenancy più tenancy meno

Il vostro SaaS prende piede: clienti in aumento, richieste di isolamento dei dati e pressione sui costi operativi. La tentazione può essere quella di dedicare un'installazione completa (Single tenancy) per ogni cliente, replicando load balancer, istanze EC2, database RDS, code di messaggistica e così via. Sebbene possa sembrare un approccio ordinato inizialmente, questo modello presenta criticità: i costi crescono

linearmente con ogni nuovo contratto, la complessità di gestione esplode (si moltiplicano gli upgrade e le patch) e i margini evaporano rapidamente.

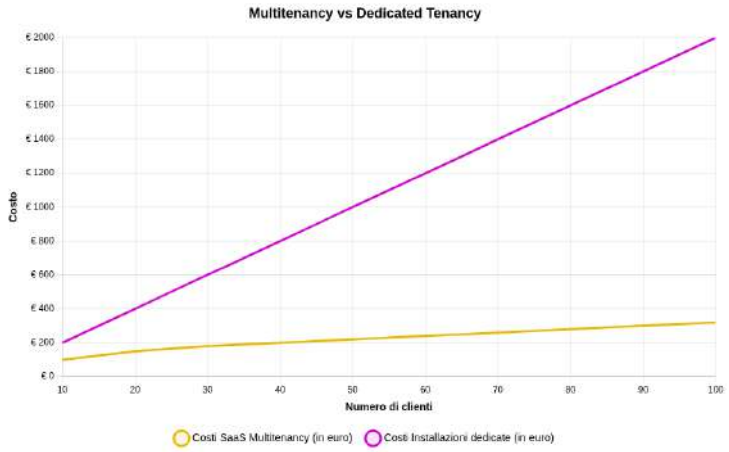
Una multi-tenancy ben progettata consente di condividere in sicurezza risorse e infrastruttura, mantenendo isolamento logico tra i clienti.

Le opzioni principali sono:

1. *Condivisione Totale*: condivisione dello strato applicativo e del database (con separazione logica per schema o tramite controlli a livello di riga).
2. *Approccio Ibrido*: applicazione condivisa con database dedicato per tenant, riservando questa opzione solo per clienti premium o con requisiti particolarmente stringenti di isolamento.

La sicurezza rimane una priorità assoluta, ma la si ottiene attraverso l'**isolamento logico** (separazione dei dati tramite codice e configurazione), policy rigorose, crittografia per tenant e un'osservabilità avanzata per prevenire i "noisy neighbors" (clienti che influenzano negativamente le prestazioni di altri), anziché duplicando l'intera infrastruttura.

Il diagramma di esempio questa volta è molto semplice. Come detto in precedenza i costi con delle installazioni single-tenant aumentano linearmente con il numero delle installazioni. Più difficile è delineare i costi dell'approccio multi-tenant, per questa analisi abbiamo considerato un incremento iniziale di circa il 50%, decrescente all'aumentare dei clienti.



I sistemi sostenibili allineano i costi alle curve di ricavo. In un'architettura multi-tenant, la crescita dei costi è molto più piatta; ogni cliente aggiuntivo porta con sé marginalità positiva anziché la necessità di mantenere una nuova "mini-piattaforma" dedicata. Evitare la clonazione dell'infrastruttura non è solo una questione di eleganza architetturale, ma è ciò che consente di scalare il business in modo sostenibile, senza far saltare il conto economico.

Tirando le somme

Gli anti-pattern che abbiamo esaminato hanno tutti un denominatore comune: nascono da decisioni prese considerando i costi come un problema "di dopo", qualcosa da affrontare quando l'applicazione è già in produzione. Ma a quel punto è troppo tardi: l'architettura è definita e modificarla diventa costoso e rischioso.

La vera lezione del FinOps è portare la consapevolezza nelle primissime fasi del ciclo di vita del software. Questo "shift-left" dei costi significa trattare l'impatto economico come un requisito alla pari di performance e sicurezza.

Come ci ricorda The Frugal Architect: "Cost is a non-functional requirement". E come ogni requisito non funzionale che si rispetti, va considerato fin dal primo giorno.

Sources:

- FinOps Framework
- FinOps Phases

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!

Andrea Pusineri

DevOps Engineer @ beSharp. Mi diverto a risolvere problemi e sono cintura nera nel trovarli. Linux enthusiast e security guy wannabe, mi piacciono le CTF e nel tempo libero sono un avido lettore di fumetti/manga/libri. btw I use Arch

---

Nicola Ferrari

Cloud Infrastructure Line Manager @ beSharp e AWS authorized instructor champion.Vivo la vita “un livello alla volta”. Ottengo i miei superpoteri raccogliendo caffeina nascosta qua e là nella mia mappa quotidiana. Sono un Internet surfer professionale (e ho visto tutto l’Internet per intero... almeno due volte!) e un appassionato di tecnologia, computer e networking. Costruire grandi cose IT - tutte precise e ordinate - contribuisce alla mia missione principale: la ricerca della perfezione!
