Halloween Special: Infrastrutture Cloud da Incubo – Episodio 3

31 Ottobre 2024 - 1 min. read

AWS Well-Architected | Cloud adoption | DevOps

"Hai visto cose orribili. Un esercito di creature da incubo. Ma non sono nulla in confronto a ciò che è venuto prima. Ciò che si trova sotto. È nostro compito placare gli Antichi. Come è tuo essere offerto a loro. Perdonaci. E facciamo presto."

The Cabin in the Woods

Ciao cari amanti dell'horror! Bentornati nel nostro piccolo angolo spaventoso!

Come molti di voi già sanno, questo periodo dell'anno è perfetto per raccontare tutte le storie spaventose su errori e problemi di design che si sono rivelati veri e propri mostri da combattere.

In questo articolo, vedremo cosa è andato storto negli ultimi 12 mesi, perché e come evitare certe cose in futuro...

Come sempre, non siamo qui a puntare il dito contro nessuno: a volte, le pessime decisioni dipendono dal contesto e dal momento... e fanno anche un po' crescere!

Quindi, sedetevi, rilassatevi (se ci riuscite) e continuate con la lettura!

Se ve li siete persi, ecco il primo e il secondo episodio.

Complessità complessa

"A volte morto è meglio."

Pet Sematary

Cosa fare quando troviamo un potenziale problema che si auto-alimenta e cresce senza che nessuno provi a fermare questo comportamento?

La nostra storia inizia durante una soleggiata giornata estiva.

Durante una Well-Architected Review, ci siamo imbattuti in un workload critico in esecuzione su EKS su EC2 ed un secondo cluster EKS basato su Fargate. Come sempre, alla ricerca di possibili miglioramenti su gestione ed elasticità, abbiamo domandato se ci fosse un piano per adottare Fargate per il workload EKS su EC2.

Non c'era nessun piano: il workload poteva girare su Fargate, ma Karpenter si occupava dello scaling del cluster. Abbiamo pensato che, comunque, fosse una cosa più che buona: Karpenter è un ottimo strumento ed è appena stato promosso uscendo dallo stato di beta. Eravamo piuttosto soddisfatti di questa scoperta perché è stata presa in considerazione la possibilità di automatizzare l'eliminazione e l'aggiunta automatica di risorse quando necessario, anche se alcuni miglioramenti potevano essere fatti pensando all'aspetto della gestibilità.

Abbiamo aperto una piccola parentesi per farci raccontare di come siano stati affrontati il tema dell'elasticità e il lavoro di ricerca e sviluppo fatto per comparare le soluzioni visto che il carico dell'applicazione era piuttosto imprevedibile. Alla fine abbiamo convenuto che fosse stato fatto un ottimo lavoro!

Abbiamo quindi dato un'occhiata al cluster EKS basato su Fargate e abbiamo scoperto che veniva utilizzato... per eseguire Karpenter!

A questo punto ci è venuto naturale domandare se il primo cluster analizzato fosse stato migrato da Autoscaler a Karpenter. Come purtroppo temevamo la risposta è stata no: avevano creato un secondo cluster per migrare l'applicazione.

Eravamo perplessi: perché non usare Fargate (o, ancora meglio, aggiungere un profilo Fargate al cluster e iniziare a migrare pod) se era necessaria una migrazione? La risposta è stata semplice semplice: "Perché usiamo Karpenter!" A questo punto eravamo ancora più confusi, ma dovevamo proseguire; abbiamo preso nota del fatto per il report (e questo articolo!).

La complessità legata all'esecuzione ed al mantenimento di uno strumento per gestire la scalabilità di risorse che in realtà possono scalare nativamente ed in modo gestito porta sicuramente a errori e problemi in futuro (soprattutto per quanto riguarda aggiornamenti e modifiche). Invece di gestire un solo componente, occorre prendersi cura di un elemento aggiuntivo e valutarne l'interazione.

In questo caso, un pattern potrebbe emergere: cosa succederebbe se in futuro venisse messo a disposizione un componente che implementa una GUI per Karpenter? Possiamo scommetterci: troveremmo la GUI in esecuzione durante la Well-Architected review! È un ciclo infinito che aggiunge complessità a un sistema per gestire la sua complessità.

Quando aggiungiamo un componente, dobbiamo valutarne l'impatto. Anche se sembra facilitare il nostro lavoro, dobbiamo sempre chiederci: c'è un servizio gestito progettato per questo?

CI/CR (Continuous Integration, Continuous Rollback)

"Le forze diaboliche sono formidabili. Queste forze sono eterne ed esistono ancora oggi. La fiaba è vera. Il diavolo esiste. Dio esiste. E per noi, come persone, il nostro stesso destino dipende da quale di essi scegliamo di seguire."

The Conjuring

"Per il nostro progetto, abbiamo bisogno di tre ambienti diversi e, ovviamente, di pipeline CI/CD per il front end ed il back end".

Niente ci rende più felici di sentire questo genere di richieste durante le call all'avvio di un progetto! Finalmente, lo scenario "Developmenttestproduction" era un lontano ricordo.

In mancanza di particolari requisiti sullo stack tecnologico, abbiamo utilizzato CodeBuild e CodePipeline e concordato di utilizzare tre branch differenti per i diversi ambienti: dev, test e production, ciascuno associato a una pipeline in esecuzione nell'ambiente specifico.

Per "promuovere" il codice tra gli ambienti, deve essere effettuato un merge da dev a test, e così via. Qualcuno potrebbe obiettare che esistono flussi migliori, ma questo sembrava

adattarsi bene al workflow di sviluppo già in uso dagli sviluppatori.

Dopo qualche tempo di sviluppo IaC con CDK, gli ambienti dev e test per frontend e backend erano pronti. A questo punto ci è stato chiesto di utilizzare l'ID del commit per identificare le release delle immagini contenute nel repository ECR e degli asset nei bucket S3. È stato un gioco da ragazzi, ma non avevamo idea di cosa stava per succedere.

Dopo un altro po' di tempo (e sviluppo), l'ambiente di produzione era pronto e il team ha richiesto un passaggio di approvazione manuale in pipeline e, successivamente, di applicare un ulteriore un git tag nel repository per identificare meglio le release. Solo dopo una settimana o due la situazione è diventata chiara: abbiamo ricevuto una richiesta di modificare le pipeline per il repository di test e produzione per poter selezionare da quale tag far partire la pipeline e fare il deploy.

Questo per poter scegliere quale versione di front end e back end distribuire in ogni ambiente per facilitare i rollback... "perché non si sa mai".

Inutile dire che questa è la strada giusta per raggiungere il disastro perfetto: fare rollback di diversi componenti può portare a interazioni sconosciute e inaspettate. Ad esempio per investigare un bug nell'ambiente di test (o, peggio, in produzione) bisognerà fare il rollback dell'intera catena negli ambienti, anche negli ambienti di sviluppo locali.

Molti articoli già parlano del costo del rollback e dei problemi che possono derivare dall'avere ambienti divergenti. In alcuni casi poi, policy aziendali rigorose vietano il deploy di versioni che non siano in esecuzione e validate nell'ambiente di test.

In tutto questo non abbiamo preso in considerazione l'errore umano: selezionare manualmente il tag sbagliato è molto più probabile rispetto all'avere una pipeline che distribuisce automaticamente l'ultima versione da un branch... Se qualcosa va storto in un deploy, poi, è più facile fare il fix e andare avanti che elaborare un workflow complicato "just in case"!

Dopo un po' di tempo e spiegazioni, abbiamo continuato a usare il workflow "normale", e indovinate un po'? Non è mai stato fatto un rollback!

Anche i cattivi piangono

"Nonostante la mia macabra reputazione, ho davvero il cuore di un bambino. Lo tengo in un barattolo sulla mia scrivania."

Robert Bloch

A volte, la sfortuna colpisce anche i cattivi.

Un'azienda ci ha contattato in emergenza dopo aver trovato il Lunedì mattina un'istanza EC2 sconosciuta. Il team IT era piccolo e quindi il check interno è stato breve: nessuno aveva avviato quella istanza, tantomeno una così grande.

La preoccupazione maggiore riguardava la possibile perdita di credenziali amministrative; abbiamo quindi iniziato ad analizzare CloudTrail per scoprire che un indirizzo IP sconosciuto da un paese straniero aveva usato una Access Key per descrivere le quota e le region in uso per poi per avviare l'istanza, scegliendo eu-west-3 (Parigi), una region non comunemente utilizzata - specialmente da utenti italiani - tranne che dal nostro cliente che, per motivi storici, l'aveva sempre utilizzata come principale (e con poche istanze).

Individuare l'anomalia è stato quindi questione di pochi minuti. Dopo qualche indagine abbiamo scoperto che l'accesso anomalo era stato fatto dai "soliti" bot, i quali avevano trovato una vulnerabilità nel framework PHP di un sito web, avevano ottenuto la chiave di accesso (che purtroppo aveva permessi troppo ampi) e usato le credenziali per avviare un cripto-miner in una regione che, per il bot, sembrava una buona idea perché non comune ed abilitata di default.

Alla fine, essere un cattivo non è un compito facile; la sfortuna può capitare a chiunque!

Riguardo al nostro nuovo cliente, dopo una revisione di emergenza, abbiamo elaborato un piano per risolvere molte criticità. Stiamo ancora lavorando su tutti i problemi, ma affidarsi alle best practice è meglio che affidarsi alla dea bendata! A proposito: avere credenziali temporanee utilizzando gli IAM roles può evitare molti mal di testa, fidatevi...

Home disautomation

> *"Sai, penso che questa cosa del Natale non sia così complicata come sembra! Ma perché dovrebbero divertirsi solo loro? Dovrebbe appartenere a chiunque! In realtà non chiunque, ma io! Perché, potrei fare un albero di Natale! E non c'è una ragione che posso trovare per cui non potrei avere un periodo natalizio! Scommetto che potrei migliorarlo anche io! Ed è esattamente quello che farò!"*
>
> **Jack Skellington, the Nightmare before Christmas**

Ehi, sono io! Sono il Damiano dal passato, ti ricordi di me? Stai prendendo in giro chi ha fatto un sacco di disastri, ma non sei perfetto. Come nerd, sei vulnerabile al richiamo delle nuove tecnologie, ansioso di provarle ad ogni opportunità, anche a casa. Quindi, sii coraggioso e racconta loro del tuo progetto di domotica che ha trasformato le luci della tua casa in veri e propri ribelli...

Ok, va bene.

Questa storia parla di come la mia pigrizia e passione per le nuove tecnologie mi abbiano tradito. Era il 2018 e in quell'anno acquistai il mio primo smart switch SonOff da usare in combinazione con Alexa per accendere una lampada nel mio soggiorno perché un interruttore della luce non era disponibile nel punto in cui lo volevo.

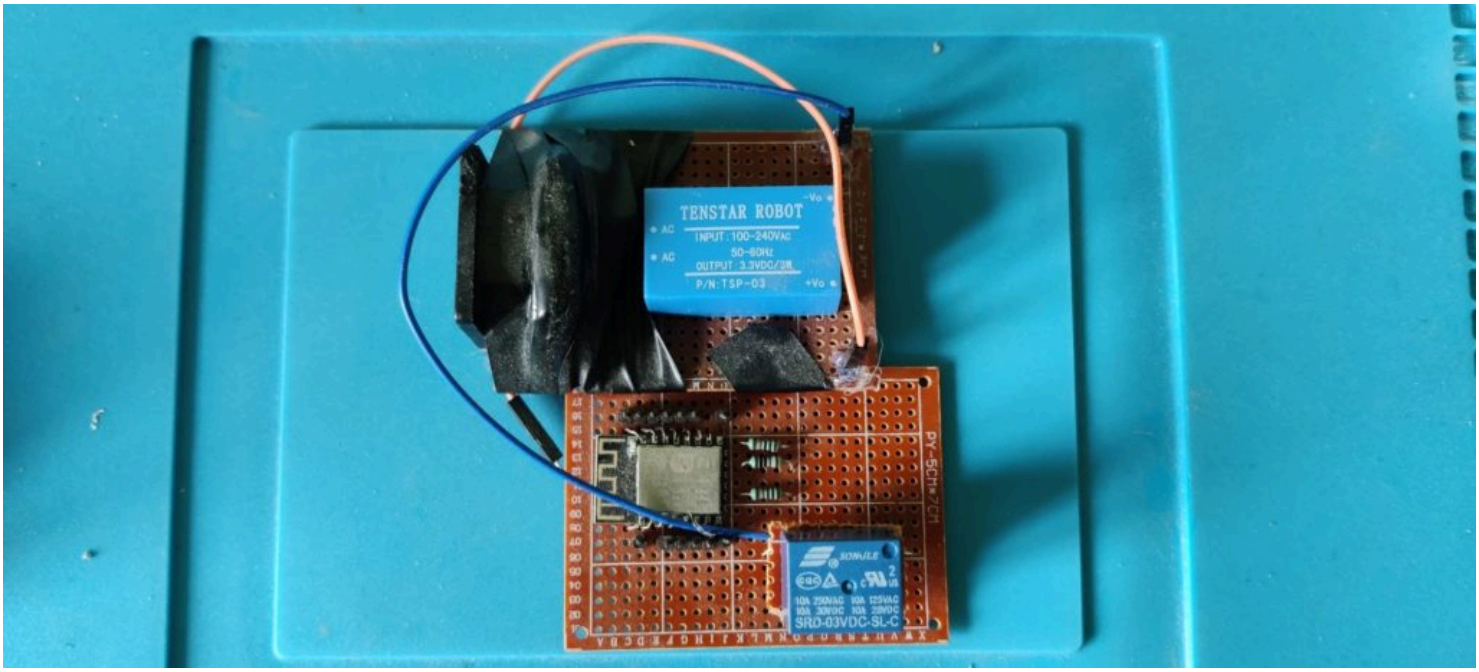
Non ci volle molto prima di scoprire la domotica con i chip Wi-Fi ESP8266, Home Assistant, Tasmota ed il protocollo MQTT. La tecnologia era nei primi anni di sviluppo e il miglior modo per imparare è sperimentare, anche con ciò che usiamo quotidianamente. Come sempre, prima di capire bene l'ordine delle cose e come funziona una tecnologia in un sistema complesso, dobbiamo fare errori.

Era Luglio, ero in vacanza in Sardegna; il telefono suonò ed all'altro capo i miei genitori mi avvisarono di una luce accesa in cucina impossibile da spegnere. Infatti, entrarono in casa mia ed invano provarono tutti i pulsanti. Sapendo che in quel periodo stavo facendo esperimenti con le luci a casa (probabilmente anche perché mio padre elettricista mi ha insegnato un po' troppe cose), sapevano già che qualcosa non era andato come previsto.

Era ovviamente colpa mia... ecco come sono andate le cose.

Avendo scoperto i chip ESP8266 e le loro capacità, ho iniziato a prototipare un "extender per pulsanti Wi-Fi" per poter controllare più lampadine con un singolo interruttore, attivando diversi eventi per diverse azioni, come ad esempio doppi clic, pressioni prolungate, ecc. L'idea era di estendere le capacità di un semplice pulsante incorporando ed aggiungendo un chip ESP8266, che potesse controllare altri dispositivi smart (anche fatti in casa) tramite webhook. In questo modo avrei potuto accendere o spegnere tutte le luci e controllare quelle in stanze differenti!

Ecco un (brutto) prototipo di quell'anno:



I problemi relativi alla luce che non si spegneva erano dovuti al fatto che:

- Usando i webhook, dovevo fare reservation DHCP sul mio router Wi-Fi e impostare l'indirizzo IP nel codice Arduino (o implementare la registrazione automatica DNS su un server locale)
- Ho quindi dimenticato di fare la reservation per l'interruttore smart
- C'è stato un blackout mentre ero in vacanza e l'indirizzo IP è ovviamente cambiato
- L'interruttore che ho usato come "extender" era, ovviamente, quello che controllava la luce.

Potevo accendere e spegnere la luce da remoto, tuttavia i miei genitori non avrebbero potuto accenderla di nuovo in caso di necessità, se non telefonandomi prima!

Cosa ho imparato da questa situazione?

1. Non cambiare il comportamento dei componenti comunemente utilizzati. Un interruttore della luce dovrebbe funzionare come un interruttore della luce. Possiamo estendere questo concetto a lambda, containers e così via; usa la tecnologia per ciò per cui è stata progettata.
2. Alcune cose, anche se "smart", dovrebbero mantenere la loro funzionalità anche se non c'è comunicazione di rete o un sistema fallisce.
3. Avere un modo per sistemare le cose (anche dalla Sardegna) è comunque bello.

"Everything fails all the time" è un motto, ma anche uno stile di vita. :D

Il malvagio piccolo punto verde

"Ho incontrato questo bambino di sei anni con questa faccia vuota, pallida, senza emozioni, e gli occhi più neri... gli occhi del diavolo. Ho passato otto anni a cercare di raggiungerlo, e poi altri sette a cercare di tenerlo rinchiuso perché ho capito che quello che viveva dietro gli occhi di quel ragazzo era puro e semplice male."

Halloween

Questa è la storia di un piccolo punto verde malvagio e di come sia riuscito a far fallire una migrazione multi-region in pochi minuti.

Era una notte tranquilla.

Dopo mesi di refactoring, containerizzazione e re-platforming, abbiamo finalmente rilasciato la nuova versione fiammante di un sistema di comunicazione distribuito che gestiva milioni di clienti e componenti.

Molte cose erano state cambiate e ottimizzate, passando da server bare metal in tutto il mondo a piccoli container in esecuzione su Fargate in diverse region con algoritmi di scaling personalizzati, da un singolo cluster MySQL a istanze MongoDB, e così via. Abbiamo

anche implementato un meccanismo di throttling delle richieste per evitare il fenomeno del "congestion collapse" (questo talk del re:Invent è assolutamente da vedere quando si ha a che fare con questo tipo di problemi).

Abbiamo ovviamente costruito una dashboard per monitorare il carico e vedere come andavano le cose. Era tardi, ma il meccanismo di throttling per mitigare il carico funzionava come previsto e tutto stava tornando alla normalità. Dopo pochi minuti, tutto ha smesso di funzionare improvvisamente e il carico sul database è aumentato.

Essendo ancora all'interno della finestra di manutenzione ed avendo la libertà di osservare le cose, abbiamo notato che i client non riuscivano più a connettersi, dando un errore dovuto al timeout sulle chiamate API e, peggio ancora, il database era usato al 100%.

Abbiamo quindi deciso di impostare il meccanismo di throttling in modo più aggressivo per capire quali componenti non si comportavano bene, ma tutto era a posto... tranne il database. Dopo 20 minuti, abbiamo chiuso il traffico dai sistemi esterni e riavviato le istanze del database. Anche questo non ha funzionato: dopo pochi minuti, il carico era tornato al 99% ed eravamo sempre più confusi.

Abbiamo iniziato l'analisi, pronti ad aprire un ticket di supporto per un'anomalia, quando abbiamo visto le query in esecuzione che saturavano l'engine del database.

Analizzando le query ne abbiamo viste tantissime del tipo db.collection.find(), e solo a quel punto abbiamo visto la luce in fondo al tunnel. La query era associata all'ultimo pezzo aggiunto i giorni prima della migrazione: si trattava di una innocua funzione Lambda associata ad API Gateway, il cui compito era riportare lo stato dei componenti del sistema. Effettuava questa semplice query:

```
for component in componentCol.find({"id":{"$in": ids}},{"id":1}):
       onlineIdsList.append(int(component["id"]))
return {"id": onlineIdsList}
```

Quell'API, che era disponibile dalla notte dei tempi sempre, era stata resa implementata per fare in modo che ogni componente potesse essere a conoscenza dello stato di tutti gli altri. Come per il resto del sistema, i dati erano stati migrati da MySQL a MongoDB. Non si sono mai verificati problemi di prestazioni in passato prima della migrazione, ma ora invece tutto andava storto, nonostante i test fossero stati effettuati. Alla fine, si trattava di una semplice query per mostrare un punto verde o rosso a seconda che il componente fosse online o non disponibile!

Come forse alcuni di voi staranno già pensando, il problema non era nella lambda o nel database. Con l'aumento dei componenti che si connettevano al nuovo sistema, il numero di entità è aumentato esponenzialmente, passando da alcune centinaia usate nei test a milioni, facendo analizzare alla query un sacco di record su database engine che non è fatto per questo tipo di attività.

La query era perfetta per un database engine relazionale, come MySQL, ma era un disastro per i database NoSQL; anche dopo alcune ottimizzazioni, abbiamo dovuto fermarci e tornare al vecchio sistema, ritardando la migrazione di almeno due settimane. Anche dopo aver fatto un refactor delle query, abbiamo dovuto usare search nodes dedicati e il lavoro è ancora in corso per portare una parte del sistema a un database relazionale, risparmiando soldi.

Cosa abbiamo imparato? Anche un semplice punto verde può essere malvagio quando si tratta di sistemi complessi, specialmente se i punti verdi, alla fine, sono milioni!

Conclusioni

Come dice il proverbio: "non c'è due senza tre"!

Il detto deriva da antiche superstizioni basate sui numeri magici che affermano che se si presentano consecutivamente due eventi della stessa natura è indubbio che in breve tempo se ne presenterà un terzo analogo, perché il Tre è il complemento del numero Magico due.

Sarà dunque questo l'ultimo episodio della nostra serie di Infrastrutture da Incubo?

Non pensiamo proprio... dato che siamo umani che affrontano problemi tecnologici. Sappiamo che in futuro troveremo nuovo materiale per questa serie, sentitevi quindi liberi di condividere le vostre storie spaventose con noi nei commenti;

magari le vedrete nel prossimo episodio!

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!

Damiano Giorgi

Ex sistemista on-prem, pigro e incline all'automazione di task noiosi. Alla ricerca costante di novità tecnologiche e quindi passato al cloud per trovare nuovi stimoli.L'unico hardware a cui mi dedico ora è quello del mio basso; se non mi trovate in ufficio o in sala prove provate al pub o in qualche aeroporto!