AWS Whitepaper

Disponibilità e oltre: comprensione e miglioramento della resilienza dei sistemi distribuiti su AWS

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Disponibilità e oltre: comprensione e miglioramento della resilienza dei sistemi distribuiti su AWS: AWS Whitepaper

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

I marchi e l'immagine commerciale di Amazon non possono essere utilizzati in relazione a prodotti o servizi che non siano di Amazon, in una qualsiasi modalità che possa causare confusione tra i clienti o in una qualsiasi modalità che denigri o discrediti Amazon. Tutti gli altri marchi non di proprietà di Amazon sono di proprietà delle rispettive aziende, che possono o meno essere associate, collegate o sponsorizzate da Amazon.

Table of Contents

Riassunto e introduzione i
Introduzione 1
Comprendere la disponibilità 2
Disponibilità distribuita del sistema 4
Tipi di guasti nei sistemi distribuiti 6
Disponibilità con dipendenze 7
Disponibilità con ridondanza 8
Teorema CAP 13
Tolleranza ai guasti e isolamento dei guasti 14
Misurazione della disponibilità 17
Percentuale di successo delle richieste lato server e lato client 17
Tempo di inattività annuale 19
Latenza 20
Progettazione di sistemi distribuiti ad alta disponibilità su AWS 22
Riducendo MTTD 22
Riducendo MTTR 23
Percorso per aggirare il fallimento 23
Ritorno a uno stato di buono stato noto 25
Diagnosi del fallimento 27
Runbook e automazione 27
Aumentando MTBF 28
Sistema distribuito in aumento MTBF 28
Dipendenza crescente MTBF 30
Ridurre le fonti di impatto comuni 31
Conclusioni 35
Appendice 1 — Metriche critiche MTTD e MTTR 37
Fattori determinanti 38
Approfondimenti 39
Cronologia dei documenti 40
Note 41
Glossario per AWS 42
..... xliii

Disponibilità e non solo: comprensione e miglioramento della resilienza dei sistemi distribuiti su AWS

Data di pubblicazione: 12 novembre 2021 (Cronologia dei documenti)

Oggi, le aziende gestiscono sistemi complessi e distribuiti sia nel cloud che in locale. Vogliono che questi carichi di lavoro siano resilienti per servire i propri clienti e raggiungere i propri risultati aziendali. Questo documento delinea una comprensione comune della disponibilità come misura della resilienza, stabilisce regole per creare carichi di lavoro ad alta disponibilità e offre indicazioni su come migliorare la disponibilità dei carichi di lavoro.

Introduzione

Cosa significa creare un carico di lavoro altamente disponibile? Come si misura la disponibilità? Cosa posso fare per aumentare la disponibilità del mio carico di lavoro? Questo documento ti aiuterà a rispondere a questo tipo di domande. È diviso in tre sezioni principali. La prima sezione, Comprendere la disponibilità, è in gran parte teorica. Stabilisce una comprensione comune della definizione di disponibilità e dei fattori che la influenzano. La seconda sezione, Misurazione della disponibilità, fornisce indicazioni sulla misurazione empirica della disponibilità del carico di lavoro. La terza sezione, Progettazione di sistemi distribuiti ad alta disponibilità su, AWS è un'applicazione pratica delle idee presentate nella prima sezione. Inoltre, in queste sezioni, questo documento identificherà le regole per la creazione di carichi di lavoro resilienti. Questo documento ha lo scopo di supportare le linee guida e le migliori pratiche presentate nel AWSWell-Architected Reliability Pillar.

In questo articolo, incontrerai molta matematica algebrica. I punti chiave sono i concetti supportati da questa matematica, non la matematica stessa. Detto questo, l'intento di questo documento è anche quello di presentare una sfida. Quando gestisci carichi di lavoro ad alta disponibilità, devi essere in grado di dimostrare, matematicamente, che ciò che hai creato sta raggiungendo gli obiettivi prefissati. Anche i migliori progetti basati su buone intenzioni potrebbero non raggiungere costantemente il risultato desiderato. Ciò significa che sono necessari meccanismi che misurino l'efficacia della soluzione e, pertanto, è necessario un certo livello di matematica per creare e gestire sistemi distribuiti resilienti e ad alta disponibilità.

Comprendere la disponibilità

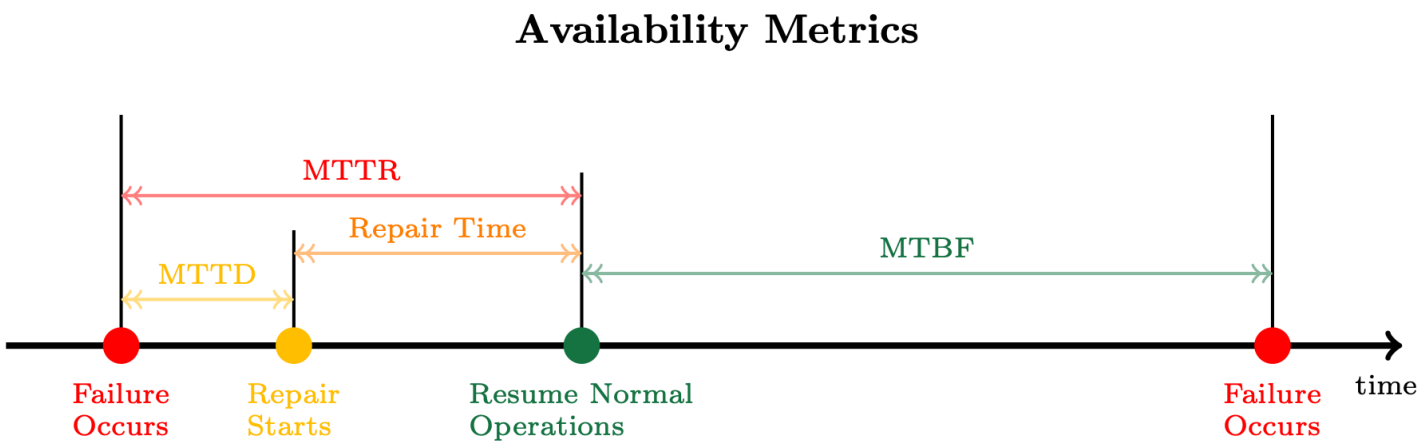
La disponibilità è uno dei modi principali in cui possiamo misurare quantitativamente la resilienza. Definiamo la disponibilità, A, come la percentuale di tempo in cui un carico di lavoro è disponibile per l'uso. È il rapporto tra il «tempo di attività» previsto (disponibilità) e il tempo totale misurato (il «tempo di attività» previsto più il «tempo di inattività» previsto).

$$A = \frac{uptime}{uptime + downtime}$$

Equazione 1 - Disponibilità

Per comprendere meglio questa formula, vedremo come misurare i tempi di attività e i tempi di inattività. Innanzitutto, vogliamo sapere per quanto tempo durerà il carico di lavoro senza errori. Lo chiamiamo Mean Time Between Failure (MTBF), il tempo medio che intercorre tra l'inizio del normale funzionamento di un carico di lavoro e il successivo guasto. Quindi, vogliamo sapere quanto tempo ci vorrà per il ripristino dopo il fallimento.

Questo periodo è denominato Mean Time to Repair (or Recovery) (MTTR), un periodo di tempo in cui il carico di lavoro non è disponibile mentre il sottosistema guasto viene riparato o rimesso in servizio. Un periodo di tempo importante nell'MTTR è il tempo medio di rilevamento (MTTD), il periodo di tempo che intercorre tra il verificarsi di un guasto e l'inizio delle operazioni di riparazione. Il diagramma seguente mostra come tutte queste metriche sono correlate.



La relazione tra MTTD, MTTR e MTBF

Possiamo quindi esprimere la disponibilità, A, utilizzando MTBF, il momento in cui il carico di lavoro è attivo e MTTR, il momento in cui il carico di lavoro è inattivo.

$$A = \frac{MTBF}{MTBF + MTTR}$$

Equazione 2 - Relazione tra MTBF e MTTR

E la probabilità che il carico di lavoro sia «inattivo» (cioè non disponibile) è la probabilità di fallimento, F.

$$F = 1 - A$$

Equazione 3 - Probabilità di fallimento

L'affidabilità è la capacità di un carico di lavoro di fare la cosa giusta, quando richiesto, entro il tempo di risposta specificato. Questo è ciò che misura la disponibilità. Un carico di lavoro che si guasta meno frequentemente (MTBF più lungo) o un tempo di riparazione più breve (MTTR più breve) ne migliora la disponibilità.

> Regola 1
>
> Guasti meno frequenti (MTBF più lungo), tempi di rilevamento dei guasti più brevi (MTTD più brevi) e tempi di riparazione più brevi (MTTR più breve) sono i tre fattori utilizzati per migliorare la disponibilità nei sistemi distribuiti.

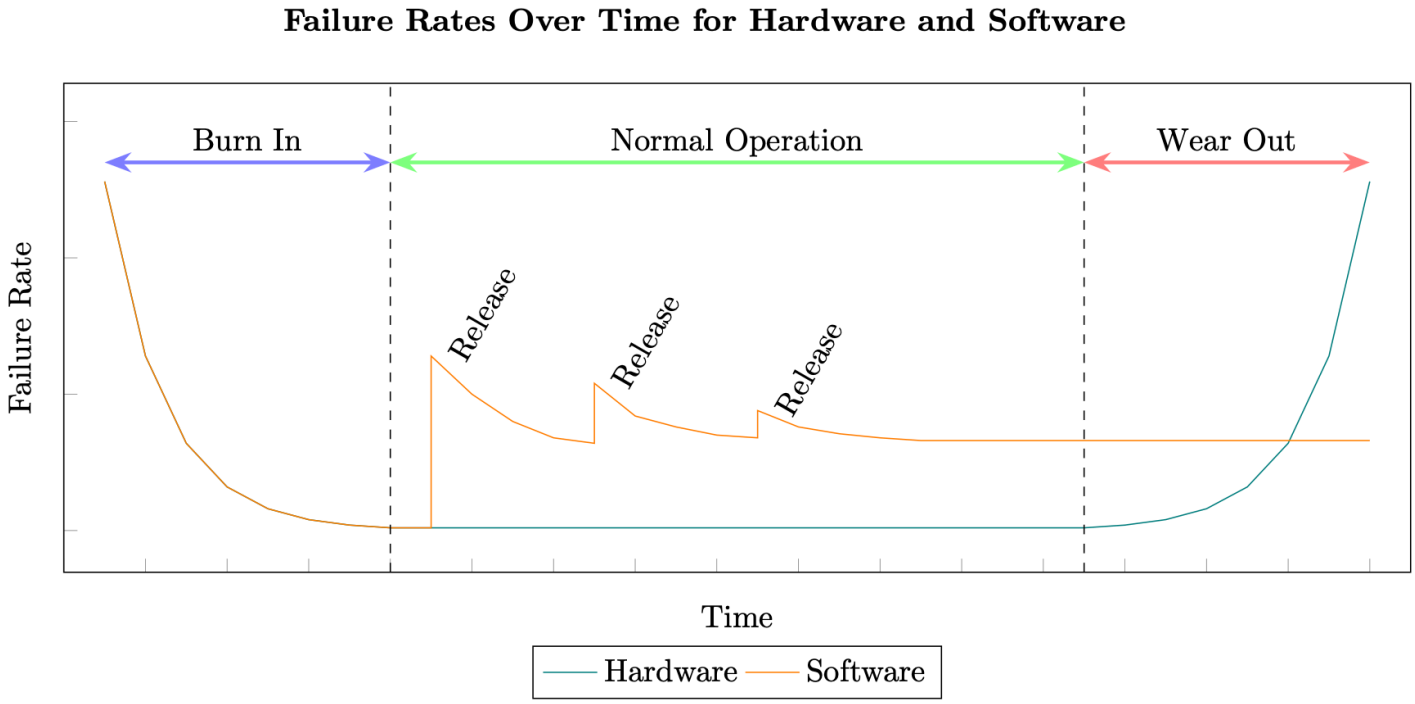
Argomenti

- Disponibilità del sistema distribuito
- Disponibilità con dipendenze
- Disponibilità con ridondanza
- Teorema CAP
- Tolleranza e isolamento dei guasti

Disponibilità del sistema distribuito

I sistemi distribuiti sono costituiti sia da componenti software che da componenti hardware. Alcuni componenti software potrebbero essere essi stessi un altro sistema distribuito. La disponibilità dei componenti hardware e software sottostanti influisce sulla conseguente disponibilità del carico di lavoro.

Il calcolo della disponibilità mediante MTBF e MTTR affonda le sue radici nei sistemi hardware. Tuttavia, i sistemi distribuiti falliscono per ragioni molto diverse rispetto a quelle di un componente hardware. Se un produttore è in grado di calcolare in modo coerente il tempo medio che precede l'usura di un componente hardware, gli stessi test non possono essere applicati ai componenti software di un sistema distribuito. L'hardware segue in genere una curva approssimativa del tasso di guasto, mentre il software segue una curva sfalsata causata da difetti aggiuntivi che vengono introdotti con ogni nuova versione (vedi Affidabilità del software).



Percentuali di guasti hardware e software

Inoltre, il software nei sistemi distribuiti cambia in genere a velocità esponenzialmente superiori a quelle dell'hardware. Ad esempio, un disco rigido magnetico standard potrebbe avere un tasso di guasto annualizzato medio (AFR) dello 0,93% che, in pratica per un HDD, può significare una durata di almeno 3-5 anni prima che raggiunga il periodo di usura, potenzialmente più lungo (vedi Backblaze Hard Drive Data and Stats, 2020). Il disco rigido non cambia sostanzialmente durante

quel ciclo di vita, dove, in 3-5 anni, ad esempio, Amazon potrebbe implementare più di 450-750 milioni di modifiche ai suoi sistemi software. (Vedi Amazon Builders' Library — Automatizzazione di distribuzioni sicure e pratiche).

L'hardware è inoltre soggetto al concetto di obsolescenza pianificata, ossia ha una durata di vita incorporata e dovrà essere sostituito dopo un certo periodo di tempo. (Vedi The Great Lightbulb Conspiracy.) Il software, in teoria, non è soggetto a questo vincolo, non ha un periodo di usura e può essere utilizzato a tempo indeterminato.

Tutto ciò significa che gli stessi modelli di test e previsione utilizzati per l'hardware per generare i numeri MTBF e MTTR non si applicano al software. Ci sono stati centinaia di tentativi di creare modelli per risolvere questo problema a partire dagli anni '70, ma generalmente rientrano tutti in due categorie: modelli di previsione e modelli di stima. (Vedi Elenco dei modelli di affidabilità del software).

Pertanto, il calcolo di un MTBF e di un MTTR lungimiranti per sistemi distribuiti, e quindi di una disponibilità previsionale, sarà sempre derivato da qualche tipo di previsione o previsione. Possono essere generati mediante modellazione predittiva, simulazione stocastica, analisi storica o test rigorosi, ma tali calcoli non garantiscono tempi di attività o tempi di inattività.

I motivi per cui un sistema distribuito ha fallito in passato potrebbero non ripresentarsi mai. Le ragioni per cui fallirà in futuro saranno probabilmente diverse e forse inconoscibili. I meccanismi di ripristino necessari potrebbero inoltre essere diversi per i guasti futuri rispetto a quelli utilizzati in passato e richiedere tempi significativamente diversi.

Inoltre, MTBF e MTTR sono valori medi. Ci sarà una certa varianza tra il valore medio e i valori effettivi visti (la deviazione standard, $_2$, misura questa variazione). Pertanto, i carichi di lavoro possono trascorrere un periodo di tempo più o meno lungo tra i guasti e i tempi di ripristino nell'effettivo utilizzo in produzione.

Detto questo, la disponibilità dei componenti software che costituiscono un sistema distribuito è ancora importante. Il software può fallire per numerose ragioni (discusse più approfonditamente nella prossima sezione) e influire sulla disponibilità del carico di lavoro. Pertanto, per i sistemi distribuiti ad alta disponibilità, occorre prestare la stessa attenzione al calcolo, alla misurazione e al miglioramento della disponibilità dei componenti software per quanto riguarda l'hardware e i sottosistemi software esterni.

> Regola 2
>
> La disponibilità del software nel carico di lavoro è un fattore importante della disponibilità complessiva del carico di lavoro e dovrebbe ricevere la stessa attenzione degli altri componenti.

È importante notare che, nonostante MTBF e MTTR siano difficili da prevedere per i sistemi distribuiti, forniscono comunque informazioni chiave su come migliorare la disponibilità. La riduzione della frequenza dei guasti (MTBF più elevato) e la riduzione del tempo di ripristino dopo il verificarsi del guasto (MTTR inferiore) porteranno entrambe a una maggiore disponibilità empirica.

Tipi di guasti nei sistemi distribuiti

Esistono generalmente due classi di bug nei sistemi distribuiti che influiscono sulla disponibilità, chiamate affettuosamente Bohrbug e Heisenbug (vedi «A Conversation with Bruce Lindsay», ACM Queue vol. 2, n. 8 — novembre 2004).

Un Bohrbug è un problema software funzionale ripetibile. Con lo stesso input, il bug produrrà costantemente lo stesso output errato (come il modello atomico deterministico di Bohr, che è solido e facilmente rilevabile). Questi tipi di bug sono rari quando un carico di lavoro entra in produzione.

Un Heisenbug è un bug transitorio, il che significa che si verifica solo in condizioni specifiche e non comuni. Queste condizioni sono in genere correlate a fattori come l'hardware (ad esempio, un guasto temporaneo del dispositivo o specifiche dell'implementazione hardware come la dimensione del registro), le ottimizzazioni del compilatore e l'implementazione del linguaggio, le condizioni limite (ad esempio, l'esaurimento temporaneo dello spazio di archiviazione) o le condizioni di gara (ad esempio, il mancato utilizzo di un semaforo per operazioni multithread).

Gli Heisenbug costituiscono la maggior parte dei bug in produzione e sono difficili da trovare perché sono sfuggenti e sembrano cambiare comportamento o scomparire quando si tenta di osservarli o eseguirne il debug. Tuttavia, se si riavvia il programma, è probabile che l'operazione fallita abbia successo perché l'ambiente operativo è leggermente diverso, eliminando le condizioni che hanno introdotto l'Heisenbug.

Pertanto, la maggior parte degli errori di produzione sono transitori e quando l'operazione viene ritentata, è improbabile che si verifichi nuovamente un errore. Per essere resilienti, i sistemi distribuiti devono essere tolleranti ai guasti nei confronti di Heisenbugs. Scopriremo come raggiungere questo obiettivo nella sezione Incrementare l'MTBF dei sistemi distribuiti.

Disponibilità con dipendenze

Nella sezione precedente, abbiamo detto che l'hardware, il software e potenzialmente altri sistemi distribuiti sono tutti componenti del carico di lavoro. Questi componenti si chiamano dipendenze, ossia gli elementi da cui dipende il carico di lavoro per fornire le proprie funzionalità. Esistono dipendenze rigide, ovvero quelle cose senza le quali il carico di lavoro non può funzionare, e dipendenze morbide la cui indisponibilità può passare inosservata o tollerata per un certo periodo di tempo. Le dipendenze rigide hanno un impatto diretto sulla disponibilità del carico di lavoro.

Potremmo provare a calcolare la disponibilità massima teorica di un carico di lavoro. Questo è il prodotto della disponibilità di tutte le dipendenze, incluso il software stesso, (α_n è la disponibilità di un singolo sottosistema) perché ognuna deve essere operativa.

$$A = \alpha_1 \times \alpha_2 \times \ldots \times \alpha_n$$

Equazione 4 - Disponibilità massima teorica

I numeri di disponibilità utilizzati in questi calcoli sono generalmente associati a elementi come SLA o Service-Level Objectives (SLO). Gli SLA definiscono il livello di servizio previsto che i clienti riceveranno, le metriche in base alle quali viene misurato il servizio e le misure correttive o le penalità (generalmente pecuniarie) in caso di mancato raggiungimento dei livelli di servizio.

Utilizzando la formula precedente, possiamo concludere che, puramente matematicamente, un carico di lavoro non può essere più disponibile di nessuna delle sue dipendenze. Ma in realtà, ciò che di solito vediamo è che non è così. Un carico di lavoro creato utilizzando due o tre dipendenze con SLA di disponibilità del 99,99% può comunque raggiungere da solo una disponibilità del 99,99% o superiore.

Questo perché, come indicato nella sezione precedente, questi numeri di disponibilità sono stime. Stimano o prevedono la frequenza con cui si verifica un guasto e la rapidità con cui può essere riparato. Non sono una garanzia di tempi di inattività. Le dipendenze spesso superano lo SLA o lo SLO di disponibilità dichiarati.

Le dipendenze possono inoltre avere obiettivi di disponibilità interna più elevati rispetto ai quali mirare alle prestazioni rispetto ai numeri forniti negli SLA pubblici. Ciò fornisce un livello di mitigazione del rischio nel rispetto degli SLA quando accade l'ignoto o l'inconoscibile.

Infine, il carico di lavoro potrebbe avere dipendenze i cui SLA non sono noti o non offrono uno SLA o uno SLO. Ad esempio, il routing Internet a livello mondiale è una dipendenza comune per molti

carichi di lavoro, ma è difficile sapere quali provider di servizi Internet utilizza il traffico globale, se hanno degli SLA e quanto siano coerenti tra i provider.

Tutto ciò ci dice che il calcolo di una disponibilità teorica massima probabilmente produce solo un calcolo approssimativo dell'ordine di grandezza, ma di per sé è probabile che non sia accurato o non fornisca informazioni significative. Ciò che la matematica ci dice è che il minor numero di elementi su cui si basa il carico di lavoro riduce la probabilità complessiva di fallimento. Minore è il numero di numeri moltiplicati tra loro, maggiore è il risultato.

> Regola 3
>
> La riduzione delle dipendenze può avere un impatto positivo sulla disponibilità.

La matematica aiuta anche a orientare il processo di selezione delle dipendenze. Il processo di selezione influisce sul modo in cui si progetta il carico di lavoro, su come si sfrutta la ridondanza delle dipendenze per migliorarne la disponibilità e sul fatto che tali dipendenze vengano considerate deboli o rigide. Le dipendenze che possono avere un impatto sul carico di lavoro devono essere scelte con attenzione. La regola successiva fornisce indicazioni su come farlo.

> Regola 4
>
> In generale, seleziona le dipendenze i cui obiettivi di disponibilità sono uguali o superiori agli obiettivi del tuo carico di lavoro.

Disponibilità con ridondanza

Quando un carico di lavoro utilizza sottosistemi multipli, indipendenti e ridondanti, può raggiungere un livello di disponibilità teorica più elevato rispetto all'utilizzo di un singolo sottosistema. Ad esempio, si consideri un carico di lavoro composto da due sottosistemi identici. Può essere completamente operativo se è operativo il sottosistema uno o il sottosistema due. Affinché l'intero sistema sia inattivo, entrambi i sottosistemi devono essere disattivati contemporaneamente.

Se la probabilità di guasto di un sottosistema è 1 − α, allora la probabilità che due sottosistemi ridondanti siano inattivi contemporaneamente è il prodotto della probabilità di guasto di ciascun sottosistema, $F = (1- \alpha) \times (1- \alpha_1)$. $_2$ Per un carico di lavoro con due sottosistemi ridondanti, utilizzando l'equazione (3), si ottiene una disponibilità definita come:

$$A = 1 - F$$
$$F = (1 - \alpha_1) \times (1 - \alpha_2)$$
$$A = 1 - (1 - \alpha)^2$$

Equazione 5

Quindi, per due sottosistemi la cui disponibilità è del 99%, la probabilità che uno fallisca è dell'1% e la probabilità che entrambi falliscano è (1− 99%) × (1− 99%) = 0,01%. Ciò rende la disponibilità utilizzando due sottosistemi ridondanti del 99,99%.

Questo può essere generalizzato per incorporare anche ricambi ridondanti aggiuntivi. Nell'Equazione (5) abbiamo ipotizzato solo un singolo pezzo di riserva, ma un carico di lavoro potrebbe avere due, tre o più pezzi di riserva in modo da poter sopravvivere alla perdita simultanea di più sottosistemi senza influire sulla disponibilità. Se un carico di lavoro ha tre sottosistemi e due sono di riserva, la probabilità che tutti e tre i sottosistemi falliscano contemporaneamente è (1− α) × (1− α) × (1− α) o (1− α) ^{3}. In generale, un carico di lavoro con s unità di riserva avrà esito negativo solo se i sottosistemi s+1 falliscono.

Per un carico di lavoro con n sottosistemi e s parti di riserva, f è il numero di modalità di errore o i modi in cui i sottosistemi s+1 possono fallire su n.

Questo è in effetti il teorema binomiale, la matematica combinatoria che consiste nello scegliere k elementi da un insieme di n, o «n sceglie k». In questo caso, k è s + 1.

$$k = s + 1$$

$$\binom{n}{k} = \frac{n!}{k!\,(n-k)!}$$

$$\binom{n}{s+1} = \frac{n!}{(s+1)!\,\big(n-(s+1)\big)!}$$

$$f = \frac{n!}{(s+1)!\,(n-s-1)!}$$

Equazione 6

Possiamo quindi produrre un'approssimazione della disponibilità generalizzata che incorpori il numero di modalità di guasto e il risparmio. (Per capire il motivo, in forma approssimativa, fate riferimento all'Appendice 2 di Highleyman, et al. Rompere la barriera della disponibilità.)

$$s = \ Number\ of\ spares$$
$$\alpha = Availability\ of\ subcomponent$$
$$f = Number\ of\ failure\ modes$$
$$A = 1 - F \approx 1 - f(1-\alpha)^{s+1}$$

Equazione 7

Lo sparing può essere applicato a qualsiasi dipendenza che fornisca risorse che falliscono indipendentemente. Ne sono un esempio le istanze Amazon EC2 in diverse AZ o i bucket Amazon S3 in diverse aree. Regioni AWS L'utilizzo delle risorse di riserva aiuta tale dipendenza a raggiungere una maggiore disponibilità totale per supportare gli obiettivi di disponibilità del carico di lavoro.

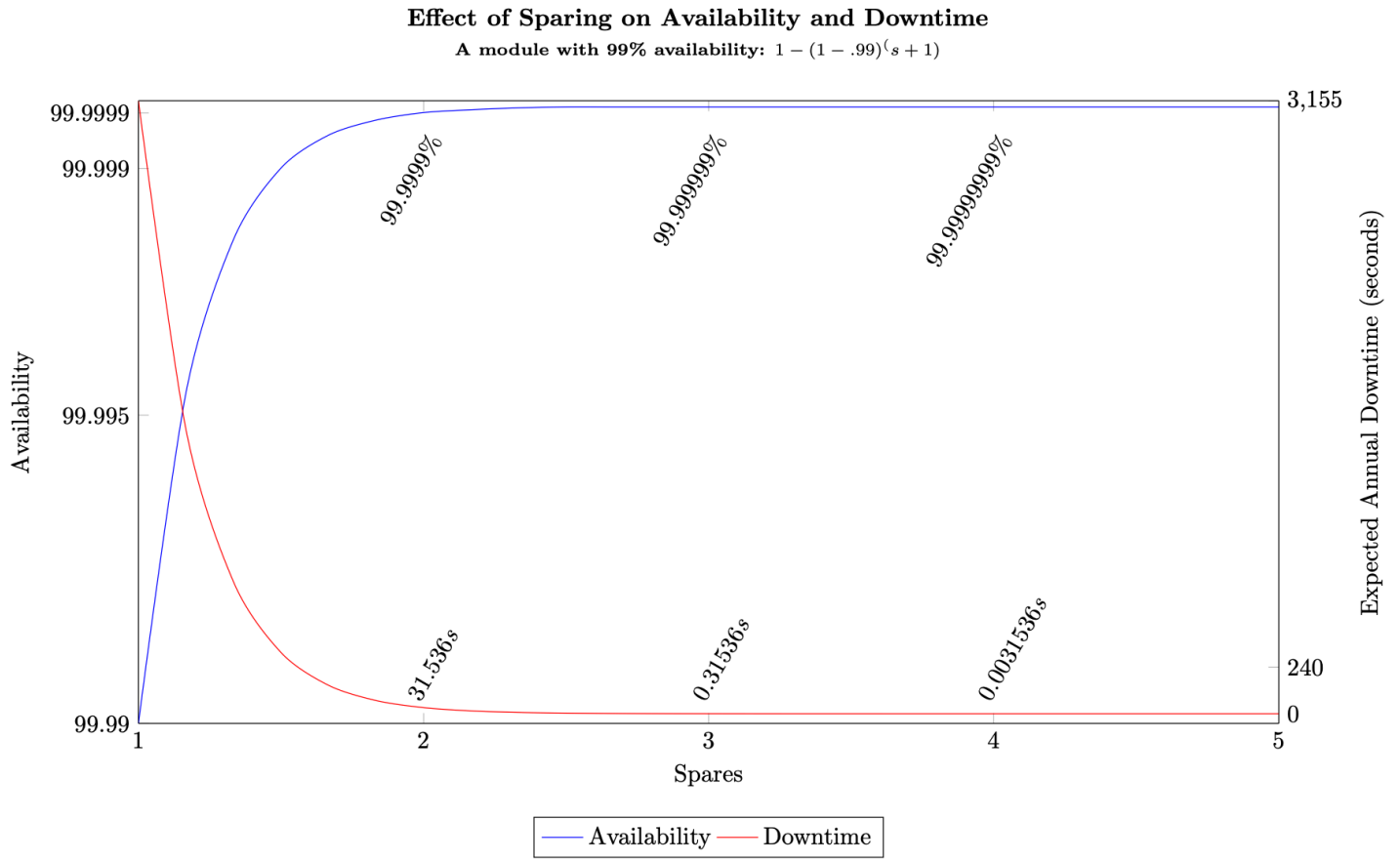
Regola 5

Usa sparing per aumentare la disponibilità delle dipendenze in un carico di lavoro.

Tuttavia, il risparmio ha un costo. Ogni pezzo di ricambio aggiuntivo ha lo stesso costo del modulo originale, con un costo almeno lineare. La creazione di un carico di lavoro in grado di utilizzare parti di ricambio ne aumenta anche la complessità. Deve saper identificare i fallimenti legati alla dipendenza, assegnare un peso al lavoro e dedicarlo a una risorsa sana e gestire la capacità complessiva del carico di lavoro.

La ridondanza è un problema di ottimizzazione. Troppi pezzi di ricambio e il carico di lavoro può fallire più frequentemente del desiderato, troppe parti di ricambio e l'esecuzione del carico di lavoro costa troppo. Esiste una soglia oltre la quale l'aggiunta di ulteriori ricambi costerà più della disponibilità aggiuntiva garantita.

Utilizzando la nostra formula di disponibilità generale con ricambi, Equation (7), per un sottosistema con una disponibilità del 99,5%, con due unità di riserva la disponibilità del carico di lavoro è $A \approx 1 - (1) (1-.995)^{3}$ = 99,9999875% (circa 3,94 secondi di inattività all'anno) e con 10 ricambi otteniamo $A \approx 1 - (1) (1-.995) = {}^{11}$ 25,5 9 ′ s (il tempo di inattività approssimativo sarebbe di $1{,}26252 \times 10^{-15}$ m s all'anno, in effetti 0). Confrontando questi due carichi di lavoro, abbiamo registrato un aumento di 5 volte del costo dello risparmio per ottenere quattro secondi di inattività in meno all'anno. Per la maggior parte dei carichi di lavoro, l'aumento dei costi sarebbe ingiustificato per questo aumento della disponibilità. La figura seguente mostra questa relazione.



Rendimenti decrescenti derivanti da un maggiore risparmio

Con tre ricambi di riserva e oltre, il risultato è una frazione di secondo dei tempi di inattività previsti all'anno, il che significa che dopo questo punto si raggiunge l'area dei rendimenti decrescenti. Potrebbe essere necessario «aggiungere altro» per raggiungere livelli di disponibilità più elevati, ma in realtà i vantaggi in termini di costi scompaiono molto rapidamente. L'utilizzo di più di tre pezzi di ricambio non offre un guadagno sostanziale e notevole per quasi tutti i carichi di lavoro quando il sottosistema stesso ha una disponibilità almeno del 99%.

> **Regola 6**
>
> Esiste un limite massimo all'efficienza in termini di costi del risparmio. Utilizzate il minor numero di ricambi necessario per ottenere la disponibilità richiesta.

È necessario considerare l'unità di guasto quando si seleziona il numero corretto di ricambi. Ad esempio, esaminiamo un carico di lavoro che richiede 10 istanze EC2 per gestire la capacità di picco e che vengono distribuite in un'unica AZ.

Poiché le AZ sono progettate per fungere da limiti di isolamento dei guasti, l'unità di errore non è solo una singola istanza EC2, poiché un'intera istanza EC2 può fallire insieme. In questo caso, ti consigliamo di aggiungere ridondanza con un'altra AZ, implementando 10 istanze EC2 aggiuntive per gestire il carico in caso di errore AZ, per un totale di 20 istanze EC2 (seguendo lo schema di stabilità statica).

Anche se sembrano essere 10 istanze EC2 di riserva, in realtà si tratta solo di una singola istanza AZ di riserva, quindi non abbiamo superato il limite di rendimenti decrescenti. Tuttavia, puoi essere ancora più efficiente in termini di costi e allo stesso tempo aumentare la disponibilità utilizzando tre AZ e implementando cinque istanze EC2 per AZ.

Ciò fornisce una zona di emergenza con un totale di 15 istanze EC2 (rispetto a due AZ con 20 istanze), fornendo comunque le 10 istanze totali necessarie per soddisfare la massima capacità durante un evento che ha un impatto su una singola AZ. Pertanto, è necessario incorporare sparing per garantire la tolleranza agli errori in tutti i limiti di isolamento dei guasti utilizzati dal carico di lavoro (istanza, cella, AZ e regione).

Teorema CAP

Un altro modo in cui potremmo pensare alla disponibilità è in relazione al teorema CAP. Il teorema afferma che un sistema distribuito, costituito da più nodi che memorizzano dati, non può fornire contemporaneamente più di due delle tre garanzie seguenti:

- Coerenza: ogni richiesta di lettura riceve la scrittura più recente o un errore quando la coerenza non può essere garantita.
- Disponibilità: ogni richiesta riceve una risposta priva di errori, anche quando i nodi sono inattivi o non disponibili.
- Tolleranza di partizione: il sistema continua a funzionare nonostante la perdita di un numero arbitrario di messaggi tra i nodi.

(Per maggiori dettagli, vedere Seth Gilbert e Nancy Lynch, La congettura di Brewer e la fattibilità di servizi web coerenti, disponibili e tolleranti alle partizioni, ACM SIGACT News, Volume 33 Numero 2 (2002), pag. 51-59.)

La maggior parte dei sistemi distribuiti deve tollerare i guasti di rete e, pertanto, deve essere consentito il partizionamento della rete. Ciò significa che questi carichi di lavoro devono scegliere tra coerenza e disponibilità quando si verifica una partizione di rete. Se il carico di lavoro sceglie la disponibilità, restituisce sempre una risposta, ma con dati potenzialmente incoerenti. Se sceglie la

coerenza, durante una partizione di rete restituirà un errore poiché il carico di lavoro non può essere sicuro della coerenza dei dati.

Per i carichi di lavoro il cui obiettivo è fornire livelli di disponibilità più elevati, potrebbero scegliere Availability and Partition Tolerance (AP) per evitare che vengano restituiti errori (non essere disponibili) durante una partizione di rete. Ciò comporta la necessità di un modello di coerenza più rilassato, come la coerenza finale o la coerenza monotona.

Tolleranza e isolamento dei guasti

Questi sono due concetti importanti quando pensiamo alla disponibilità. La tolleranza ai guasti è la capacità di resistere ai guasti del sottosistema e di mantenere la disponibilità (facendo la cosa giusta entro uno SLA stabilito). Per implementare la tolleranza agli errori, i carichi di lavoro utilizzano sottosistemi di riserva (o ridondanti). Quando uno dei sottosistemi di un set ridondante si guasta, un altro riprende il lavoro, in genere quasi senza interruzioni. In questo caso, i pezzi di ricambio rappresentano una vera e propria capacità inutilizzata, in quanto sono in grado di assorbire il 100% del lavoro del sottosistema guasto. Con True Spares, sono necessari diversi guasti del sottosistema per avere un impatto negativo sul carico di lavoro.

L'isolamento dei guasti riduce al minimo la portata dell'impatto in caso di guasto. Questo viene in genere implementato con la modularizzazione. I carichi di lavoro sono suddivisi in piccoli sottosistemi che si guastano indipendentemente e possono essere riparati in modo isolato. L'errore di un modulo non si propaga oltre il modulo. Questa idea si estende sia verticalmente, attraverso funzionalità diverse in un carico di lavoro, sia orizzontalmente, su più sottosistemi che forniscono le stesse funzionalità. Questi moduli fungono da contenitori di guasti che limitano l'ambito dell'impatto durante un evento.

I modelli architettonici dei piani di controllo, dei piani dati e della stabilità statica supportano direttamente l'implementazione della tolleranza e dell'isolamento dei guasti. L'articolo della Amazon Builders' Library Stability using Availability Zones fornisce buone definizioni per questi termini e per come si applicano alla creazione di carichi di lavoro resilienti e ad alta disponibilità. Questo white paper utilizza questi modelli nella sezione Progettazione di sistemi distribuiti ad alta disponibilità suAWS, e qui riassumiamo anche le relative definizioni.

- Piano di controllo: la parte del carico di lavoro coinvolta nell'apportare modifiche: aggiunta di risorse, eliminazione di risorse, modifica delle risorse e propagazione di tali modifiche dove sono necessarie. I piani di controllo sono in genere più complessi e hanno più parti mobili rispetto ai piani dati, quindi statisticamente hanno maggiori probabilità di guasti e hanno una disponibilità inferiore.

- Piano dati: la parte del carico di lavoro che fornisce le day-to-day funzionalità aziendali. I piani dati tendono ad essere più semplici e funzionano a volumi più elevati rispetto ai piani di controllo, con conseguente maggiore disponibilità.
- Stabilità statica: la capacità di un carico di lavoro di continuare a funzionare correttamente nonostante la riduzione della dipendenza. Un metodo di implementazione consiste nel rimuovere le dipendenze dei piani di controllo dai piani dati. Un altro metodo consiste nell'accoppiare liberamente le dipendenze del carico di lavoro. Forse il carico di lavoro non vede alcuna informazione aggiornata (come elementi nuovi, elementi eliminati o elementi modificati) che la sua dipendenza avrebbe dovuto fornire. Tuttavia, tutto ciò che faceva prima che la dipendenza diventasse compromessa continua a funzionare.

Quando pensiamo alla riduzione del carico di lavoro, ci sono due approcci di alto livello che possiamo prendere in considerazione per il recupero. Il primo metodo consiste nel reagire a tale deterioramento dopo che si è verificato, magari aggiungendo nuova capacità. AWS Auto Scaling Il secondo metodo consiste nel prepararsi a tali problemi prima che si verifichino, magari sovradimensionando l'infrastruttura di un carico di lavoro in modo che possa continuare a funzionare correttamente senza la necessità di risorse aggiuntive.

Un sistema staticamente stabile utilizza quest'ultimo approccio. Predispone la capacità inutilizzata per renderla disponibile in caso di guasto. Questo metodo evita di creare una dipendenza da un piano di controllo nel percorso di ripristino del carico di lavoro per fornire nuova capacità di ripristino in caso di guasto. Inoltre, il provisioning di nuova capacità per varie risorse richiede tempo. In attesa di una nuova capacità, il carico di lavoro può essere sovraccaricato dalla domanda esistente e subire un ulteriore degrado, con conseguente «esaurimento» o completa perdita di disponibilità. Tuttavia, è necessario considerare anche le implicazioni in termini di costi dell'utilizzo della capacità preimpostata rispetto agli obiettivi di disponibilità.

La stabilità statica fornisce le due regole successive per i carichi di lavoro ad alta disponibilità.

> Regola 7
>
> Non fatevi dipendere dai piani di controllo del vostro piano dati, specialmente durante il ripristino.

Regola 8

Associa liberamente le dipendenze in modo che il carico di lavoro possa funzionare correttamente nonostante la riduzione della dipendenza, ove possibile.

Misurazione della disponibilità

Come abbiamo visto in precedenza, creare un modello di disponibilità lungimirante per un sistema distribuito è difficile e potrebbe non fornire le informazioni desiderate. Ciò che può fornire maggiore utilità è lo sviluppo di metodi coerenti per misurare la disponibilità del carico di lavoro.

La definizione di disponibilità come uptime e downtime rappresenta un errore come opzione binaria, a prescindere dal fatto che il carico di lavoro sia attivo o meno.

Tuttavia, questo è raramente il caso. L'errore ha un certo impatto e si verifica spesso in alcuni sottoinsiemi del carico di lavoro, influendo su una percentuale di utenti o richieste, su una percentuale di sedi o su un percentile di latenza. Queste sono tutte modalità di guasto parziale.

E sebbene MTTR e MTBF siano utili per comprendere cosa determina la disponibilità risultante di un sistema e, quindi, come migliorarlo, la loro utilità non è una misura empirica della disponibilità. Inoltre, i carichi di lavoro sono composti da molti componenti. Ad esempio, un carico di lavoro come un sistema di elaborazione dei pagamenti è costituito da molte interfacce di programmazione delle applicazioni (API) e sottosistemi. Quindi, quando vogliamo porre una domanda del tipo «qual è la disponibilità dell'intero carico di lavoro?» , in realtà è una domanda complessa e ricca di sfumature.

In questa sezione, esamineremo tre modi in cui è possibile misurare empiricamente la disponibilità: percentuale di successo delle richieste lato server, percentuale di successo delle richieste lato client e tempi di inattività annuali.

Percentuale di successo delle richieste lato server e lato client

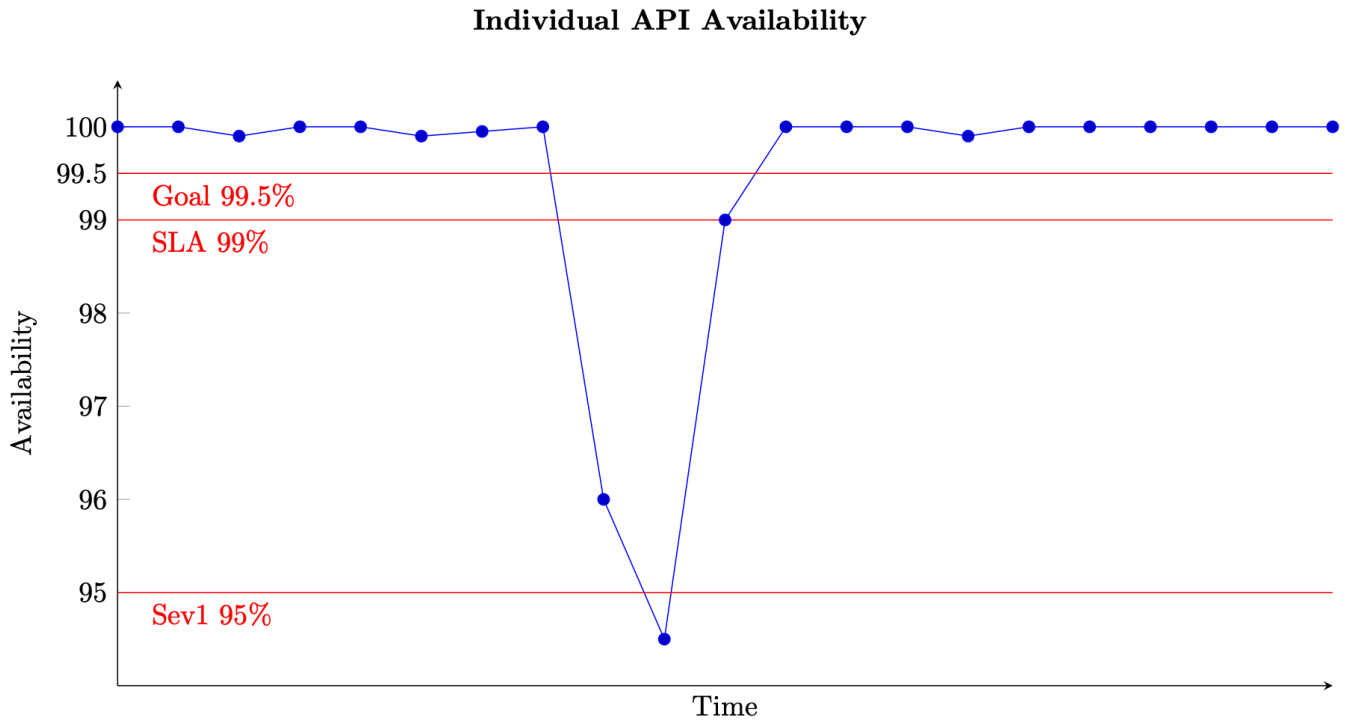
I primi due metodi sono molto simili, differiscono solo dal punto di vista della misurazione effettuata. Le metriche lato server possono essere raccolte dalla strumentazione del servizio. Tuttavia, non sono completi. Se i clienti non sono in grado di accedere al servizio, non puoi raccogliere tali metriche. Per comprendere l'esperienza del cliente, invece di affidarsi alla telemetria dei clienti per le richieste non riuscite, un modo più semplice per raccogliere dati sul lato client consiste nel simulare il traffico dei clienti con canarini, un software che analizza regolarmente i servizi e registra le metriche.

Questi due metodi calcolano la disponibilità come la frazione del totale delle unità di lavoro valide che il servizio riceve e quelle elaborate correttamente (questo ignora le unità di lavoro non valide, come una richiesta HTTP che genera un errore 404).

$$A = \frac{Successfully\ Processed\ Units\ of\ Work}{Total\ Valid\ Units\ of\ Work\ Received}$$

Equazione 8

Per un servizio basato su richieste, l'unità di lavoro è la richiesta, come una richiesta HTTP. Per i servizi basati su eventi o attività, le unità di lavoro sono eventi o attività, come l'elaborazione di un messaggio da una coda. Questa misura della disponibilità è utile in brevi intervalli di tempo, come finestre di un minuto o cinque minuti. È anche più adatto da una prospettiva granulare, ad esempio a livello di API per un servizio basato su richiesta. La figura seguente fornisce una panoramica di come potrebbe apparire la disponibilità nel tempo se calcolata in questo modo. Ogni data point sul grafico viene calcolato utilizzando l'equazione (8) in una finestra di cinque minuti (puoi scegliere altre dimensioni temporali come intervalli di un minuto o dieci minuti). Ad esempio, il punto dati 10 mostra una disponibilità del 94,5%. Ciò significa che, nei minuti da t+45 a t+50, se il servizio ha ricevuto 1.000 richieste, solo 945 di esse sono state elaborate correttamente.



Esempio di misurazione della disponibilità nel tempo per una singola API

Il grafico mostra anche l'obiettivo di disponibilità dell'API, la disponibilità del 99,5%, il contratto sul livello di servizio (SLA) offerto ai clienti, la disponibilità del 99% e la soglia per un allarme ad alta

gravità, il 95%. Senza il contesto di queste diverse soglie, un grafico della disponibilità potrebbe non fornire informazioni significative sul funzionamento del servizio.

Vogliamo anche essere in grado di tracciare e descrivere la disponibilità di un sottosistema più grande, come un piano di controllo o un intero servizio. Un modo per farlo consiste nel calcolare la media di ogni data point di cinque minuti per ogni sottosistema. Il grafico sarà simile a quello precedente, ma sarà rappresentativo di un insieme più ampio di input. Assegna inoltre lo stesso peso a tutti i sottosistemi che costituiscono il servizio. Un approccio alternativo potrebbe consistere nel sommare tutte le richieste ricevute ed elaborate con successo da tutte le API del servizio per calcolare la disponibilità a intervalli di cinque minuti.

Tuttavia, quest'ultimo metodo potrebbe nascondere una singola API con una velocità effettiva ridotta e una scarsa disponibilità. Come semplice esempio, considera un servizio con due API.

La prima API riceve 1.000.000 di richieste in una finestra di cinque minuti e ne elabora con successo 999.000, con una disponibilità del 99,9%. La seconda API riceve 100 richieste nella stessa finestra di cinque minuti e ne elabora correttamente solo 50, garantendo una disponibilità del 50%.

Se sommiamo le richieste di ciascuna API, ci sono 1.000.100 richieste valide totali e 999.050 di esse vengono elaborate correttamente, con una disponibilità complessiva del servizio del 99,895%. Tuttavia, se calcoliamo la media delle disponibilità delle due API, secondo il primo metodo, otteniamo una disponibilità del 74,95%, che potrebbe essere più indicativa dell'esperienza effettiva.

Nessuno dei due approcci è sbagliato, ma dimostra l'importanza di capire cosa ci dicono le metriche di disponibilità. Puoi scegliere di preferire la somma delle richieste per tutti i sottosistemi se il tuo carico di lavoro riceve un volume di richieste simile per ognuno di essi. Questo approccio si concentra sulla «richiesta» e sul suo successo come misura della disponibilità e dell'esperienza del cliente. In alternativa, è possibile scegliere di calcolare la disponibilità media dei sottosistemi per rappresentarne equamente la criticità nonostante le differenze nel volume delle richieste. Questo approccio si concentra sul sottosistema e sulla capacità di ciascuno di essi come indicatore dell'esperienza del cliente.

Tempo di inattività annuale

Il terzo approccio consiste nel calcolare i tempi di inattività annuali. Questa forma di metrica della disponibilità è più appropriata per la definizione e la revisione degli obiettivi a lungo termine. È necessario definire il significato dei tempi di inattività per il carico di lavoro. È quindi possibile misurare la disponibilità in base al numero di minuti in cui il carico di lavoro non è stato in condizioni di «interruzione» rispetto al numero totale di minuti in un determinato periodo.

Alcuni carichi di lavoro potrebbero essere in grado di definire il downtime come una riduzione della disponibilità al di sotto del 95% di una singola API o funzione di carico di lavoro per un intervallo di un minuto o cinque minuti (come nel precedente grafico della disponibilità). È inoltre possibile considerare i tempi di inattività solo in quanto si applicano a un sottoinsieme di operazioni critiche sul piano dati. Ad esempio, il contratto sul livello di servizio di Amazon Messaging (SQS, SNS) per la disponibilità di SQS si applica all'API di invio, ricezione ed eliminazione di SQS.

Carichi di lavoro più grandi e complessi potrebbero dover definire metriche di disponibilità a livello di sistema. Per un sito di e-commerce di grandi dimensioni, una metrica a livello di sistema può essere qualcosa come la percentuale degli ordini dei clienti. In questo caso, un calo del 10% o più degli ordini rispetto alla quantità prevista durante qualsiasi finestra di cinque minuti può determinare i tempi di inattività.

In entrambi gli approcci, è quindi possibile sommare tutti i periodi di interruzione per calcolare una disponibilità annuale. Ad esempio, se durante un anno solare si sono verificati 27 periodi di inattività da cinque minuti, definiti come la disponibilità di qualsiasi API del piano dati che scende al di sotto del 95%, il tempo di inattività complessivo è stato di 135 minuti (alcuni periodi di cinque minuti potrebbero essere stati consecutivi, altri isolati), con una disponibilità annua del 99,97%.

Questo metodo aggiuntivo di misurazione della disponibilità può fornire dati e informazioni mancanti nelle metriche lato client e lato server. Ad esempio, si consideri un carico di lavoro ridotto e con tassi di errore significativamente elevati. I clienti con questo carico di lavoro potrebbero smettere del tutto di effettuare chiamate ai suoi servizi. Forse hanno attivato un interruttore automatico o hanno seguito il piano di disaster recovery per utilizzare il servizio in un'altra regione. Se misurassimo solo le risposte fallite, la disponibilità del carico di lavoro può effettivamente aumentare durante la riduzione, ma non perché la riduzione migliora o scompare, ma perché i clienti semplicemente smettono di utilizzarlo.

Latenza

Infine, è anche importante misurare la latenza delle unità di lavoro di elaborazione all'interno del carico di lavoro. Parte della definizione di disponibilità consiste nell'eseguire il lavoro nell'ambito di uno SLA stabilito. Se la restituzione di una risposta richiede più tempo del timeout del client, il client ha la percezione che la richiesta sia fallita e il carico di lavoro non sia disponibile. Tuttavia, sul lato server, la richiesta potrebbe sembrare essere stata elaborata correttamente.

La misurazione della latenza fornisce un'altra lente con cui valutare la disponibilità. L'uso dei percentili e della media ridotta sono buone statistiche per questa misurazione. Sono comunemente

misurati al 50° percentile (P50 e TM50) e al 99° percentile (P99 e TM99). La latenza deve essere misurata con canarini per rappresentare l'esperienza del cliente e con metriche lato server. Ogni volta che la media di una latenza percentile, come P99 o TM99.9, supera lo SLA target, puoi considerare il tempo di inattività, che contribuisce al calcolo dei tempi di inattività annuali.

Progettazione di sistemi distribuiti ad alta disponibilità su AWS

Le sezioni precedenti hanno riguardato principalmente la disponibilità teorica dei carichi di lavoro e i risultati che possono ottenere. Si tratta di un insieme importante di concetti da tenere a mente quando si creano sistemi distribuiti. Ti aiuteranno a orientare il processo di selezione delle dipendenze e a come implementare la ridondanza.

Abbiamo anche esaminato la relazione e MTBF la MTTD MTTR disponibilità. Questa sezione introdurrà una guida pratica basata sulla teoria precedente. In breve, i carichi di lavoro ingegneristici per l'alta disponibilità mirano ad aumentare MTBF e diminuire il MTTRMTTD.

Sebbene l'eliminazione di tutti i guasti sarebbe l'ideale, non è realistico. Nei sistemi distribuiti di grandi dimensioni con dipendenze fortemente impilate, si verificheranno dei guasti. «Tutto fallisce sempre» (vedi Werner Vogels, Amazon.comCTO, 10 lezioni da 10 anni di Amazon Web Services.) e «non puoi legiferare contro i fallimenti [quindi] concentrati su un rilevamento e una risposta rapidi». (vedi Chris Pinkham, membro fondatore del EC2 team Amazon, ARC335Designing for failure: Architecting resilient systems on) AWS

Ciò significa che spesso non si ha il controllo sull'eventualità che si verifichi un guasto. Ciò che puoi controllare è la rapidità con cui rilevi l'errore e fai qualcosa al riguardo. Pertanto, sebbene l'aumento MTBF sia ancora una componente importante dell'elevata disponibilità, i cambiamenti più importanti che i clienti possono controllare sono la riduzione MTTD e la riduzioneMTTR.

Argomenti

- Ridurre MTTD
- Ridurre MTTR
- In aumento MTBF

Ridurre MTTD

Ridurre la probabilità MTTD di un guasto significa scoprirlo il più rapidamente possibile. L'abbreviazione si MTTD basa sull'osservabilità o sul modo in cui hai strumentato il tuo carico di lavoro per comprenderne lo stato. I clienti devono monitorare le metriche relative all'esperienza dei clienti nei sottosistemi critici del carico di lavoro per identificare in modo proattivo quando si verifica un problema (consulta l'Appendice 1) MTTD e MTTR le metriche critiche per ulteriori informazioni su

queste metriche.). I clienti possono utilizzare Amazon CloudWatch Synthetics per creare canarini che monitorano le APIs tue console e misurare in modo proattivo l'esperienza utente. Esistono diversi altri meccanismi di controllo dello stato che possono essere utilizzati per ridurli al minimoMTTD, come i controlli di integrità di Elastic Load Balancing (ELB), i controlli di integrità di Amazon Route 53 e altro ancora. (Vedi Amazon Builders' Library — Implementazione dei controlli sanitari.)

Il monitoraggio deve inoltre essere in grado di rilevare guasti parziali sia del sistema nel suo insieme che nei singoli sottosistemi. Le metriche di disponibilità, guasto e latenza devono utilizzare la dimensionalità dei limiti di isolamento dei guasti come dimensioni metriche. CloudWatch Ad esempio, si consideri una singola EC2 istanza che fa parte di un'architettura basata su celle, in use1-az1 AZ, nella regione us-east-1, che fa parte dell'aggiornamento API del carico di lavoro che fa parte del relativo sottosistema del piano di controllo. Quando il server inserisce le proprie metriche, può utilizzare l'id dell'istanza, AZ, la regione, il nome e il nome del sottosistema come dimensioni. API Ciò consente di garantire l'osservabilità e di impostare allarmi su ciascuna di queste dimensioni per rilevare i guasti.

Ridurre MTTR

Dopo l'individuazione di un guasto, il MTTR tempo restante è dedicato all'effettiva riparazione o mitigazione dell'impatto. Per riparare o mitigare un guasto, è necessario sapere cosa c'è che non va. Esistono due gruppi chiave di metriche che forniscono informazioni dettagliate durante questa fase: 1/ metriche di Impact Assessment e 2/ Metriche Operational Health. Il primo gruppo indica l'ambito dell'impatto durante un guasto, misurando il numero o la percentuale di clienti, risorse o carichi di lavoro interessati. Il secondo gruppo aiuta a identificare il motivo dell'impatto. Dopo aver scoperto il motivo, gli operatori e l'automazione possono rispondere e risolvere il problema. Per ulteriori informazioni su queste metriche, consulta l'Appendice 1 MTTD e le metriche MTTR critiche.

> Regola 9
>
> L'osservabilità e la strumentazione sono fondamentali per ridurre e. MTTD MTTR

Percorso per aggirare il fallimento

L'approccio più rapido per mitigare l'impatto consiste nell'utilizzare sottosistemi fail-fast che evitino il guasto. Questo approccio utilizza la ridondanza per ridurre il problema MTTR spostando rapidamente il lavoro di un sottosistema guasto su un sottosistema di riserva. La ridondanza può variare da processi software, a EC2 istanze, a più o meno regioni. AZs

I sottosistemi di riserva possono MTTR ridurla quasi a zero. Il tempo di ripristino è solo quello necessario per reindirizzare il lavoro alla riserva di stand-by. Ciò avviene spesso con una latenza minima e consente di completare il lavoro entro i limiti definitiSLA, mantenendo la disponibilità del sistema. MTTRsCiò produce ritardi percepiti come lievi, forse anche impercettibili, piuttosto che periodi prolungati di indisponibilità.

Ad esempio, se il servizio utilizza EC2 istanze basate su un Application Load Balancer ALB (), è possibile configurare i controlli di integrità a intervalli di soli cinque secondi e richiedere solo due controlli di integrità non riusciti prima che un oggetto venga contrassegnato come non integro. Ciò significa che entro 10 secondi è possibile rilevare un errore e interrompere l'invio di traffico all'host non funzionante. In questo caso, MTTR è effettivamente la stessa, MTTD poiché non appena viene rilevato l'errore, anche questo viene mitigato.

Questo è ciò che i carichi di lavoro ad alta disponibilità o disponibilità continua stanno cercando di ottenere. Vogliamo ovviare rapidamente ai guasti del carico di lavoro rilevando rapidamente i sottosistemi guasti, contrassegnandoli come guasti, interrompendo l'invio di traffico verso di essi e inviando invece il traffico a un sottosistema ridondante.

Tieni presente che l'utilizzo di questo tipo di meccanismo rapido rende il carico di lavoro molto sensibile agli errori transitori. Nell'esempio fornito, assicurati che i controlli sullo stato del bilanciamento del carico eseguano controlli superficiali o attivi e locali solo sull'istanza, senza testare le dipendenze o i flussi di lavoro (spesso denominati controlli approfonditi dello stato). Ciò contribuirà a prevenire la sostituzione non necessaria delle istanze in caso di errori transitori che influiscono sul carico di lavoro.

L'osservabilità e la capacità di rilevare i guasti nei sottosistemi sono fondamentali per il corretto instradamento del sistema. È necessario conoscere la portata dell'impatto in modo che le risorse interessate possano essere contrassegnate come non integre o guaste e disattivate e messe fuori servizio in modo da poter essere instradate. Ad esempio, se una singola AZ subisce un'interruzione parziale del servizio, la strumentazione dovrà essere in grado di identificare l'esistenza di un problema localizzato in AZ che riguarda il routing di tutte le risorse in quella zona fino al ripristino.

La capacità di aggirare i guasti potrebbe inoltre richiedere strumenti aggiuntivi a seconda dell'ambiente. Utilizzando l'esempio precedente con EC2 le istanze che si basano su unaALB, immaginate che le istanze di una zona possano superare i controlli sanitari locali, ma che un problema isolato della AZ impedisca loro di connettersi al database in un'altra AZ. In questo caso, i controlli di integrità del bilanciamento del carico non metteranno fuori servizio tali istanze. Sarebbe necessario un meccanismo automatizzato diverso per rimuovere l'AZ dal sistema di bilanciamento del

carico o costringere le istanze a non superare i controlli di integrità, il che dipende dall'identificazione dell'ambito di impatto di una AZ. Per i carichi di lavoro che non utilizzano un sistema di bilanciamento del carico, sarebbe necessario un metodo simile per evitare che le risorse di una zona specifica accettino unità di lavoro o sottraggano del tutto capacità alla zona AZ.

In alcuni casi, il trasferimento del lavoro verso un sottosistema ridondante non può essere automatizzato, come il failover di un database primario su un database secondario in cui la tecnologia non prevede la scelta del proprio leader. Si tratta di uno scenario comune per le architetture multiregionali.AWS Poiché questi tipi di failover richiedono una certa quantità di downtime per essere eseguiti, non possono essere immediatamente annullati e lasciano il carico di lavoro senza ridondanza per un periodo di tempo, è importante avere un operatore umano nel processo decisionale.

I carichi di lavoro che possono adottare un modello di coerenza meno rigoroso possono ridursi utilizzando l'automazione del failover multiregionale per aggirare i guasti. MTTRs Funzionalità come la replica interregionale di Amazon S3 o le tabelle globali di Amazon DynamoDB forniscono funzionalità multiregionali attraverso una replica alla fine coerente. Inoltre, l'utilizzo di un modello di coerenza rilassato è utile se consideriamo il teorema. CAP Durante i guasti di rete che influiscono sulla connettività ai sottosistemi stateful, se il carico di lavoro preferisce la disponibilità alla coerenza, può comunque fornire risposte prive di errori, un altro modo per aggirare il problema.

Il routing in caso di guasto può essere implementato con due strategie diverse. La prima strategia consiste nell'implementare la stabilità statica predisponendo risorse sufficienti per gestire il carico completo del sottosistema guasto. Può trattarsi di una singola EC2 istanza o di un'intera AZ di capacità. Il tentativo di fornire nuove risorse durante un errore aumenta MTTR e aggiunge una dipendenza da un piano di controllo nel percorso di ripristino. Tuttavia, comporta un costo aggiuntivo.

La seconda strategia consiste nell'instradare parte del traffico dal sottosistema guasto ad altri sottosistemi e caricare il traffico in eccesso che non può essere gestito dalla capacità residua. Durante questo periodo di degrado, è possibile aumentare le nuove risorse per sostituire la capacità guasta. Questo approccio è più lungo MTTR e crea una dipendenza da un piano di controllo, ma costa meno in caso di capacità inutilizzata e in standby.

Ritorno a uno stato di buono stato conosciuto

Un altro approccio comune per la mitigazione durante la riparazione consiste nel riportare il carico di lavoro al precedente buono stato noto. Se una modifica recente potrebbe aver causato l'errore, ripristinare tale modifica è un modo per tornare allo stato precedente.

In altri casi, l'errore potrebbe essere stato causato da condizioni transitorie, nel qual caso il riavvio del carico di lavoro potrebbe mitigare l'impatto. Esaminiamo entrambi questi scenari.

Durante un'implementazione, la riduzione al minimo dell'energia MTTR si basa sull'osservabilità MTTD e sull'automazione. Il processo di implementazione deve monitorare costantemente il carico di lavoro per verificare l'introduzione di un aumento dei tassi di errore, di una maggiore latenza o di anomalie. Una volta riconosciuti, dovrebbe arrestare il processo di distribuzione.

Esistono varie strategie di implementazione, come le implementazioni sul posto, le implementazioni blu/verdi e le implementazioni continue. Ognuno di questi potrebbe utilizzare un meccanismo diverso per tornare a uno stato di buono stato riconosciuto. Può tornare automaticamente allo stato precedente, riportare il traffico all'ambiente blu o richiedere un intervento manuale.

CloudFormation offre la possibilità di eseguire automaticamente il rollback come parte delle operazioni di creazione e aggiornamento dello stack, così come fa. AWS CodeDeploy CodeDeploy supporta anche implementazioni blu/green e rotative.

Per sfruttare queste funzionalità e ridurre al minimo le tueMTTR, prendi in considerazione l'automazione di tutta l'infrastruttura e le implementazioni del codice tramite questi servizi. Negli scenari in cui non è possibile utilizzare questi servizi, prendi in considerazione l'implementazione del modello Saga AWS Step Functions per ripristinare le distribuzioni non riuscite.

Quando si considera il riavvio, esistono diversi approcci. Questi vanno dal riavvio di un server, l'operazione più lunga, al riavvio di un thread, l'operazione più breve. Ecco una tabella che descrive alcuni degli approcci di riavvio e i tempi approssimativi di completamento (rappresentativi della differenza di ordini di grandezza, non sono esatti).

Meccanismo di ripristino degli errori	Stimato MTTR
Avvia e configura un nuovo server virtuale	15 minuti
Ridistribuisci il software	10 minuti
Riavviare il server	5 minuti
Riavvia o avvia il contenitore	2 secondi
Invoca una nuova funzione serverless	100 ms

Meccanismo di ripristino degli errori	Stimato MTTR
Processo di riavvio	10 ms
Riavvia il thread	10 µs

Esaminando la tabella, ci sono alcuni chiari vantaggi derivanti dall'uso di contenitori e funzioni serverless (come AWS Lambda). MTTR Sono ordini MTTR di grandezza più rapidi rispetto al riavvio di una macchina virtuale o al lancio di una nuova macchina virtuale. Tuttavia, anche l'utilizzo dell'isolamento dei guasti tramite la modularità del software è vantaggioso. Se è possibile contenere gli errori di un singolo processo o thread, il ripristino da tale errore è molto più rapido rispetto al riavvio di un contenitore o di un server.

Come approccio generale al ripristino, è possibile passare dal basso verso l'alto: 1/Restart, 2/Reboot, 3/Re-image/Redeploy, 4/Replace. Tuttavia, una volta che si arriva alla fase di riavvio, l'instradamento in caso di guasto è in genere un approccio più rapido (in genere richiede al massimo 3-4 minuti). Quindi, per mitigare più rapidamente l'impatto dopo un tentativo di riavvio, aggira l'errore e poi, in background, continua il processo di ripristino per ripristinare la capacità del carico di lavoro.

> Regola 10
>
> Concentrati sulla mitigazione degli impatti, non sulla risoluzione dei problemi. Riprendi il percorso più rapido per tornare alla normale operatività.

Diagnosi dell'errore

Parte del processo di riparazione dopo il rilevamento è il periodo di diagnosi. Questo è il periodo di tempo in cui gli operatori cercano di determinare cosa c'è che non va. Questo processo potrebbe comportare l'interrogazione dei log, la revisione delle metriche di Operational Health o l'accesso agli host per la risoluzione dei problemi. Tutte queste azioni richiedono tempo, quindi la creazione di strumenti e runbook per velocizzarle può contribuire anche a ridurle. MTTR

Runbook e automazione

Analogamente, dopo aver determinato cosa non va e quale linea di condotta consentirà di riparare il carico di lavoro, gli operatori in genere devono eseguire una serie di passaggi per eseguire questa operazione. Ad esempio, dopo un errore, il modo più rapido per riparare il carico di lavoro potrebbe

essere riavviarlo, operazione che può comportare più passaggi ordinati. L'utilizzo di un runbook che automatizzi questi passaggi o fornisca istruzioni specifiche a un operatore velocizzerà il processo e contribuirà a ridurre il rischio di azioni involontarie.

In aumento MTBF

L'ultimo componente per migliorare la disponibilità è l'aumento diMTBF. Ciò può applicarsi sia al software che ai AWS servizi utilizzati per eseguirlo.

Aumento del sistema distribuito MTBF

Un modo per aumentare MTBF è ridurre i difetti del software. Esistono vari modi per eseguire questa operazione. I clienti possono utilizzare strumenti come Amazon CodeGuru Reviewer per trovare e correggere errori comuni. È inoltre necessario eseguire revisioni complete del codice tra pari, test unitari, test di integrazione, test di regressione e test di carico sul software prima che venga distribuito in produzione. L'aumento della copertura del codice nei test contribuirà a garantire che vengano testati anche i percorsi di esecuzione del codice non comuni.

L'implementazione di modifiche minori può anche aiutare a prevenire risultati imprevisti riducendo la complessità delle modifiche. Ogni attività offre l'opportunità di identificare e correggere i difetti prima che possano essere invocati.

Un altro approccio per prevenire i guasti è costituito dai test regolari. L'implementazione di un programma di ingegneria del caos può aiutare a verificare gli errori del carico di lavoro, a convalidare le procedure di ripristino e a individuare e correggere le modalità di errore prima che si verifichino in produzione. I clienti possono utilizzare AWS Fault Injection Simulator come parte del loro set di strumenti per esperimenti di ingegneria del caos.

La tolleranza ai guasti è un altro modo per prevenire i guasti in un sistema distribuito. Moduli fail-fast, nuovi tentativi con backoff e jitter esponenziali, transazioni e idempotenza sono tutte tecniche che contribuiscono a rendere i carichi di lavoro tolleranti ai guasti.

Le transazioni sono un gruppo di operazioni che aderiscono alle proprietà. ACID Essi sono i seguenti:

- Atomicità: o tutte le azioni si verificano o nessuna di esse accadrà.
- Coerenza: ogni transazione lascia il carico di lavoro in uno stato valido.
- Isolamento: le transazioni eseguite contemporaneamente lasciano il carico di lavoro nello stesso stato come se fossero state eseguite in sequenza.

- Durabilità: una volta completata la transazione, tutti i suoi effetti vengono preservati anche in caso di errore del carico di lavoro.

I nuovi tentativi con backoff e jitter esponenziali consentono di superare i guasti transitori causati da Heisenbug, sovraccarico o altre condizioni. Quando le transazioni sono idempotenti, possono essere ritentate più volte senza effetti collaterali.

Se consideriamo l'effetto di un Heisenbug su una configurazione hardware tollerante ai guasti, non saremmo affatto preoccupati, poiché la probabilità che l'Heisenbug compaia sia sul sottosistema primario che su quello ridondante è infinitesimale. (Vedi Jim Gray, "Perché i computer si fermano e cosa si può fare al riguardo? «, giugno 1985, Tandem Technical Report 85.7.) Nei sistemi distribuiti, vogliamo ottenere gli stessi risultati con il nostro software.

Quando viene richiamato un Heisenbug, è fondamentale che il software rilevi rapidamente l'operazione errata e fallisca in modo che possa essere riprovato. Ciò si ottiene attraverso una programmazione difensiva e la convalida degli input, dei risultati intermedi e dell'output. Inoltre, i processi sono isolati e non condividono lo stato con altri processi.

Questo approccio modulare garantisce che l'ambito di impatto in caso di guasto sia limitato. I processi falliscono indipendentemente. Quando un processo fallisce, il software deve utilizzare delle «coppie di processi» per riprovare a eseguire il lavoro, il che significa che un nuovo processo può assumere il funzionamento di un processo fallito. Per mantenere l'affidabilità e l'integrità del carico di lavoro, ogni operazione deve essere trattata come una transazione. ACID

Ciò consente il fallimento di un processo senza alterare lo stato del carico di lavoro, interrompendo la transazione e ripristinando le modifiche apportate. Ciò consente al processo di ripristino di ritentare la transazione da uno stato riconosciuto valido e di riavviarla correttamente. Questo è il modo in cui il software può essere tollerante ai guasti nei confronti di Heisenbugs.

Tuttavia, non dovresti mirare a rendere il software tollerante ai guasti nei confronti di Bohrbugs. Questi difetti devono essere individuati e rimossi prima che il carico di lavoro entri in produzione, poiché nessun livello di ridondanza porterà mai a risultati corretti. (Vedi Jim Gray, "Perché i computer si fermano e cosa si può fare al riguardo? «, giugno 1985, Tandem Technical Report 85.7.)

L'ultimo modo per aumentare MTBF è ridurre la portata dell'impatto derivante da un guasto. L'utilizzo dell'isolamento dei guasti mediante la modularizzazione per creare contenitori di guasti è un modo fondamentale per farlo, come illustrato in precedenza in Tolleranza agli errori e isolamento dei guasti. La riduzione del tasso di errore migliora la disponibilità. AWS utilizza tecniche come la suddivisione dei servizi in piani di controllo e piani dati, l'indipendenza dalla zona di disponibilità (AZI), l'isolamento

regionale, le architetture basate su celle e lo shuffle-sharding per fornire l'isolamento dai guasti. Questi sono modelli che possono essere utilizzati anche dai clienti. AWS

Ad esempio, esaminiamo uno scenario in cui un carico di lavoro ha collocato i clienti in diversi contenitori di guasto della sua infrastruttura che servivano al massimo il 5% del totale dei clienti. In uno di questi contenitori di errore si verifica un evento che ha aumentato la latenza oltre il timeout del client per il 10% delle richieste. Durante questo evento, per il 95% dei clienti, il servizio era disponibile al 100%. Per il restante 5%, il servizio sembrava disponibile al 90%. Ciò si traduce in una disponibilità di 1 − (5% o f c u s t o m e r s × 10% o f t h e i r r e q u e s t s) = 99,5% anziché il 10% delle richieste non riuscite per il 100% dei clienti (con una disponibilità del 90%).

> Regola 11
>
> L'isolamento dei guasti riduce la portata dell'impatto e aumenta il carico MTBF di lavoro riducendo il tasso di guasto complessivo.

Aumento della dipendenza MTBF

Il primo metodo per aumentare la AWS dipendenza MTBF consiste nell'utilizzare l'isolamento dei guasti. Molti AWS servizi offrono un livello di isolamento in una AZ, il che significa che un guasto in una AZ non influisce sul servizio in un'altra AZ.

L'utilizzo di istanze ridondanti in più EC2 istanze AZs aumenta la disponibilità del sottosistema. AZIoffre una funzionalità di risparmio all'interno di una singola regione, che consente di aumentare la disponibilità dei servizi. AZI

Tuttavia, non tutti i AWS servizi funzionano a livello di AZ. Molti altri offrono l'isolamento regionale. In questo caso, se la disponibilità prevista del servizio regionale non supporta la disponibilità complessiva richiesta per il carico di lavoro, potresti prendere in considerazione un approccio multiregionale. Ogni regione offre un'istanza isolata del servizio, equivalente a sparing.

Esistono vari servizi che aiutano a semplificare la creazione di un servizio multiregionale. Per esempio:

- Database globale Amazon Aurora
- Tabelle globali Amazon DynamoDB
- Amazon ElastiCache (RedisOSS) — Datastore globale
- AWS Acceleratore globale

- Replica tra regioni Amazon S3
- Controller di ripristino delle applicazioni Amazon Route 53

Questo documento non approfondisce le strategie di creazione di carichi di lavoro multiregionali, ma è necessario valutare i vantaggi in termini di disponibilità delle architetture multiregionali con i costi aggiuntivi, la complessità e le pratiche operative necessarie per raggiungere gli obiettivi di disponibilità desiderati.

Il metodo successivo per aumentare la dipendenza consiste nel progettare il carico di lavoro in modo che MTBF sia staticamente stabile. Ad esempio, hai un carico di lavoro che serve informazioni sul prodotto. Quando i tuoi clienti richiedono un prodotto, il tuo servizio invia una richiesta a un servizio di metadati esterno per recuperare i dettagli del prodotto. Quindi il carico di lavoro restituisce tutte queste informazioni all'utente.

Tuttavia, se il servizio di metadati non è disponibile, le richieste effettuate dai clienti hanno esito negativo. Puoi invece estrarre o inviare localmente i metadati in modo asincrono al tuo servizio in modo asincrono per utilizzarli per rispondere alle richieste. In questo modo si elimina la chiamata sincrona al servizio di metadati dal percorso critico.

Inoltre, poiché il servizio è ancora disponibile anche quando il servizio di metadati non lo è, puoi rimuoverlo come dipendenza nel calcolo della disponibilità. Questo esempio si basa sul presupposto che i metadati non vengano modificati frequentemente e che la pubblicazione di metadati obsoleti sia preferibile all'esito negativo della richiesta. Un altro esempio simile è serve-stale for DNS che consente di conservare i dati nella cache oltre la TTL scadenza e di utilizzarli per le risposte quando una risposta aggiornata non è prontamente disponibile.

L'ultimo metodo per aumentare la dipendenza consiste nel ridurre la portata dell'impatto MTBF derivante da un errore. Come accennato in precedenza, il fallimento non è un evento binario, ma vi sono diversi gradi di fallimento. Questo è l'effetto della modularizzazione; l'errore è limitato solo alle richieste o agli utenti serviti da quel contenitore.

Ciò si traduce in un minor numero di errori durante un evento, il che in ultima analisi aumenta la disponibilità del carico di lavoro complessivo limitando l'ambito dell'impatto.

Riduzione delle fonti di impatto comuni

Nel 1985, Jim Gray scoprì, durante uno studio presso Tandem Computers, che il fallimento era causato principalmente da due fattori: il software e le operazioni. (Vedi Jim Gray, "Perché i computer si fermano e cosa si può fare al riguardo? «, giugno 1985, Tandem Technical Report 85.7.) Anche

dopo 36 anni, questo continua ad essere vero. Nonostante i progressi tecnologici, non esiste una soluzione semplice a questi problemi e le principali fonti di fallimento non sono cambiate. La risoluzione dei guasti nel software è stata discussa all'inizio di questa sezione, quindi l'attenzione sarà rivolta alle operazioni e alla riduzione della frequenza dei guasti.

Stabilità rispetto alle funzionalità

Se facciamo riferimento al grafico dei tassi di errore per software e hardware nella sezionethe section called “Disponibilità distribuita del sistema”, possiamo notare che i difetti vengono aggiunti in ogni versione del software. Ciò significa che qualsiasi modifica al carico di lavoro comporta un aumento del rischio di fallimento. Queste modifiche sono in genere cose come nuove funzionalità, che ne forniscono un corollario. Una maggiore disponibilità dei carichi di lavoro favorirà la stabilità rispetto alle nuove funzionalità. Pertanto, uno dei modi più semplici per migliorare la disponibilità consiste nell'implementare meno spesso o fornire meno funzionalità. I carichi di lavoro che vengono implementati più frequentemente avranno intrinsecamente una disponibilità inferiore rispetto a quelli che non lo fanno. Tuttavia, i carichi di lavoro che non aggiungono funzionalità non tengono il passo con la domanda dei clienti e possono diventare meno utili nel tempo.

Quindi, come possiamo continuare a innovare e rilasciare funzionalità in modo sicuro? La risposta è la standardizzazione. Qual è il modo corretto di implementazione? Come si ordinano le distribuzioni? Quali sono gli standard per i test? Quanto tempo aspetti tra una fase e l'altra? I vostri test unitari coprono una parte sufficiente del codice software? Si tratta di domande a cui la standardizzazione può rispondere e prevenire problemi causati da fattori come il mancato test di carico, il salto delle fasi di implementazione o l'implementazione troppo rapida su troppi host.

Il modo in cui si implementa la standardizzazione avviene attraverso l'automazione. Riduce la possibilità di errori umani e consente ai computer di fare ciò in cui sono bravi, ovvero fare la stessa cosa più e più volte nello stesso modo ogni volta. Il modo per unire standardizzazione e automazione consiste nel fissare degli obiettivi. Obiettivi come l'assenza di modifiche manuali, l'accesso all'host solo tramite sistemi di autorizzazione contingente, la stesura di test di carico per tutti API e così via. L'eccellenza operativa è una norma culturale che può richiedere cambiamenti sostanziali. Stabilire e monitorare le prestazioni rispetto a un obiettivo aiuta a promuovere un cambiamento culturale che avrà un ampio impatto sulla disponibilità del carico di lavoro. Il pilastro AWS Well-Architected Operational Excellence fornisce best practice complete per l'eccellenza operativa.

Sicurezza degli operatori

L'altro fattore importante che contribuisce agli eventi operativi che determinano il fallimento sono le persone. Gli esseri umani commettono errori. Potrebbero utilizzare le credenziali sbagliate,

immettere il comando sbagliato, premere Invio troppo presto o perdere un passaggio fondamentale. L'esecuzione di azioni manuali comporta costantemente errori, con conseguente fallimento.

Una delle cause principali degli errori degli operatori sono le interfacce utente confuse, poco intuitive o incoerenti. Jim Gray ha anche osservato nel suo studio del 1985 che «le interfacce che richiedono informazioni all'operatore o gli chiedono di eseguire alcune funzioni devono essere semplici, coerenti e tolleranti ai guasti dell'operatore». (Vedi Jim Gray, "Perché i computer si fermano e cosa si può fare al riguardo? «, giugno 1985, Tandem Technical Report 85.7.) Questa intuizione continua ad essere vera oggi. Negli ultimi tre decenni in tutto il settore ci sono numerosi esempi in cui un'interfaccia utente confusa o complessa, la mancanza di conferme o istruzioni o anche solo un linguaggio umano ostile hanno indotto un operatore a fare la cosa sbagliata.

> Regola 12
>
> Fai in modo che gli operatori facciano facilmente la cosa giusta.

Prevenzione del sovraccarico

L'ultimo fattore comune che contribuisce all'impatto sono i vostri clienti, gli utenti effettivi del vostro carico di lavoro. I carichi di lavoro efficaci tendono a essere utilizzati spesso, ma a volte tale utilizzo supera la capacità di scalabilità del carico di lavoro. Ci sono molte cose che possono succedere, i dischi possono riempirsi, i pool di thread potrebbero esaurirsi, la larghezza di banda della rete potrebbe essere saturata o si possono raggiungere i limiti di connessione al database.

Non esiste un metodo infallibile per eliminarli, ma il monitoraggio proattivo della capacità e dell'utilizzo tramite le metriche di Operational Health fornirà avvisi tempestivi quando potrebbero verificarsi questi guasti. Tecniche come la riduzione del carico, gli interruttori automatici e i nuovi tentativi con backoff e jitter esponenziali possono contribuire a ridurre al minimo l'impatto e aumentare la percentuale di successo, ma queste situazioni rappresentano comunque un fallimento. La scalabilità automatizzata basata su metriche di Operational Health può aiutare a ridurre la frequenza dei guasti dovuti al sovraccarico, ma potrebbe non essere in grado di rispondere abbastanza rapidamente ai cambiamenti di utilizzo.

Se è necessario garantire la disponibilità continua della capacità per i clienti, è necessario scendere a compromessi in termini di disponibilità e costi. Un modo per garantire che la mancanza di capacità non porti all'indisponibilità è fornire a ciascun cliente una quota e garantire che la capacità del carico di lavoro sia scalata in modo da fornire il 100% delle quote assegnate. Quando i clienti superano la

loro quota, vengono limitati, il che non è un problema e non influisce sulla disponibilità. Dovrai inoltre monitorare attentamente la tua base clienti e prevedere l'utilizzo futuro per mantenere una capacità sufficiente. Ciò garantisce che il carico di lavoro non sia ricondotto a scenari di errore dovuti a un consumo eccessivo da parte dei clienti.

- Amazon Builders' Library: utilizzo del load shedding per evitare il sovraccarico
- Amazon Builders' Library: equità nei sistemi multi-tenant

Ad esempio, esaminiamo un carico di lavoro che fornisce un servizio di storage. Ogni server del carico di lavoro può supportare 100 download al secondo, ai clienti viene fornita una quota di 200 download al secondo e i clienti sono 500. Per poter supportare questo volume di clienti, il servizio deve fornire una capacità di 100.000 download al secondo, il che richiede 1.000 server. Se un cliente supera la propria quota, viene limitato, il che garantisce una capacità sufficiente per tutti gli altri clienti. Questo è un semplice esempio di un modo per evitare il sovraccarico senza scartare le unità di lavoro.

Conclusioni

Abbiamo stabilito 12 regole per l'alta disponibilità in questo documento.

- Regola 1 — Guasti meno frequenti (MTBF più lungo), tempi di rilevamento dei guasti più brevi (MTTD più breve) e tempi di riparazione più brevi (MTTR più breve) sono i tre fattori utilizzati per migliorare la disponibilità nei sistemi distribuiti.
- Regola 2 — La disponibilità del software nel carico di lavoro è un fattore importante della disponibilità complessiva del carico di lavoro e dovrebbe ricevere la stessa attenzione degli altri componenti.
- Regola 3 — La riduzione delle dipendenze può avere un impatto positivo sulla disponibilità.
- Regola 4: in generale, seleziona le dipendenze i cui obiettivi di disponibilità sono uguali o superiori agli obiettivi del tuo carico di lavoro.
- Regola 5: usa il risparmio per aumentare la disponibilità delle dipendenze in un carico di lavoro.
- Regola 6 — Esiste un limite superiore all'efficienza in termini di costi del risparmio. Utilizza il minor numero di pezzi di ricambio necessari per ottenere la disponibilità richiesta.
- Regola 7: non dipendere dai piani di controllo del piano dati, specialmente durante il ripristino.
- Regola 8 — Abbina liberamente le dipendenze in modo che il carico di lavoro possa funzionare correttamente nonostante la riduzione della dipendenza, ove possibile.
- Regola 9 — L'osservabilità e la strumentazione sono fondamentali per ridurre MTTD e MTTR.
- Regola 10 — Concentrati sulla mitigazione dell'impatto, non sulla risoluzione dei problemi. Prendi il percorso più veloce per tornare al normale funzionamento.
- Regola 11 — L'isolamento dei guasti riduce la portata dell'impatto e aumenta l'MTBF del carico di lavoro riducendo il tasso di guasto complessivo.
- Regola 12 — Fai in modo che gli operatori facciano facilmente la cosa giusta.

Il miglioramento della disponibilità dei carichi di lavoro si basa sulla riduzione di MTTD e MTTR e sull'aumento del MTBF. In sintesi, abbiamo discusso i seguenti modi per migliorare la disponibilità che riguardano tecnologia, persone e processi.

- MTTD
  - Riduci l'MTTD attraverso il monitoraggio proattivo delle metriche relative alla Customer Experience.

  - Sfrutta i controlli di integrità granulari per un failover rapido.
- MTTR
  - Monitora l'ambito di impatto e le metriche operative sullo stato di salute.
  - Riduci l'MTTR seguendo 1/Restart, 2/Reboot, 3/Re-image/Redeploy e 4/Replace.
  - Risolvi il fallimento comprendendo l'ambito dell'impatto.
  - Utilizza servizi con tempi di riavvio più rapidi, come container e funzioni serverless su macchine virtuali o host fisici.
  - Ripristina automaticamente le distribuzioni non riuscite quando possibile.
  - Definisci guide e strumenti operativi per le operazioni di diagnosi e le procedure di riavvio.
- MTBF
  - Elimina i bug e i difetti del software attraverso test rigorosi prima del rilascio in produzione.
  - Implementa l'ingegneria del caos e l'iniezione di errori.
  - Utilizza la giusta dose di risparmio nelle dipendenze per tollerare il fallimento.
  - Riduci al minimo la portata dell'impatto durante i guasti tramite contenitori di guasti.
  - Implementa gli standard per le implementazioni e le modifiche.
  - Progetta interfacce operatore semplici, intuitive, coerenti e ben documentate.
  - Stabilisci obiettivi per l'eccellenza operativa.
  - Privilegia la stabilità rispetto al rilascio di nuove funzionalità quando la disponibilità è una dimensione critica del carico di lavoro.
  - Implementa le quote di utilizzo con limitazione o riduzione del carico o entrambi per evitare il sovraccarico.

Ricorda che non riusciremo mai completamente a prevenire i fallimenti. Concentrati sulla progettazione di software con il miglior isolamento possibile dei guasti che limiti la portata e l'entità dell'impatto, idealmente mantenendo tale impatto al di sotto delle soglie di «downtime» E investi in un rilevamento e una mitigazione molto rapidi e affidabili. I moderni sistemi distribuiti devono ancora considerare i guasti come inevitabili ed essere progettati a tutti i livelli per garantire un'elevata disponibilità.

Appendice 1 — Metriche critiche MTTD e MTTR

Quello che segue è un framework per la standardizzazione della strumentazione e dell'osservabilità che può aiutare a ridurre l'MTTD e l'MTTR durante un evento.

Metriche sull'esperienza del cliente. Queste metriche indicano che un servizio è reattivo e disponibile per soddisfare le richieste dei clienti. Ad esempio, la latenza del piano di controllo. Queste metriche misurano il tasso di errore, la disponibilità, la latenza, il volume e la velocità di accelerazione.

Metriche di valutazione dell'impatto. Queste metriche forniscono informazioni sull'ambito dell'impatto durante gli eventi. Ad esempio, il numero o la percentuale di clienti interessati da un evento del piano dati. Misura il numero o la percentuale di elementi interessati.

Metriche sanitarie operative. Queste metriche indicano che un servizio è reattivo e disponibile per soddisfare le richieste dei clienti, ma si concentra su sottosistemi e risorse dell'infrastruttura comuni. Ad esempio, la percentuale di utilizzo della CPU della tua flotta EC2. Queste metriche devono misurare l'utilizzo, la capacità, la velocità effettiva, il tasso di errore, la disponibilità e la latenza.

Fattori determinanti

I contributori a questo documento includono:

- Michael Haken, Principal Solutions Architect, Amazon Web Services

Approfondimenti

Per ulteriori informazioni, fare riferimento a:

- Pilastro dell'affidabilità ben progettato
- Pilastro dell'eccellenza operativa ben architettato
- Amazon Builders' Library: garantire la sicurezza del rollback durante le implementazioni
- Amazon Builders' Library — Oltre i cinque nove: lezioni dai nostri piani dati più elevati disponibili
- Amazon Builders' Library: automazione di implementazioni sicure e pratiche
- Amazon Builders' Library: progettazione e gestione di sistemi serverless resilienti su larga scala
- Amazon Builders' Library: l'approccio di Amazon all'implementazione ad alta disponibilità
- Amazon Builders' Library: l'approccio di Amazon alla creazione di servizi resilienti
- Amazon Builders' Library: l'approccio di Amazon al fallimento
- AWSCentro di architettura

Cronologia dei documenti

Per ricevere notifiche sugli aggiornamenti di questo white paper, iscriviti al feed RSS.

Modifica	Descrizione	Data
Pubblicazione iniziale	Whitepaper pubblicato per la prima volta.	12 novembre 2021

> **Note**
>
> Per sottoscrivere gli aggiornamenti RSS, devi avere un plug-in RSS abilitato per il browser che stai utilizzando.

Note

I clienti sono responsabili della propria valutazione indipendente delle informazioni contenute in questo documento. Questo documento: (a) è solo a scopo informativo, (b) rappresenta le attuali offerte e pratiche di AWS prodotti, che sono soggette a modifiche senza preavviso e (c) non crea alcun impegno o garanzia da parte AWS e dei suoi affiliati, fornitori o licenzianti. AWSi prodotti o i servizi sono forniti «così come sono» senza garanzie, dichiarazioni o condizioni di alcun tipo, espresse o implicite. Le responsabilità e le responsabilità nei AWS confronti dei propri clienti sono controllate da AWS accordi e il presente documento non fa parte, né modifica, alcun accordo tra AWS e i suoi clienti.

© 2021 Amazon Web Services, Inc. o società affiliate. Tutti i diritti riservati.

Glossario per AWS

Per la terminologia AWS più recente, consultare il glossario AWS nella documentazione di riferimento per Glossario AWS.

Le traduzioni sono generate tramite traduzione automatica. In caso di conflitto tra il contenuto di una traduzione e la versione originale in Inglese, quest'ultima prevarrà.