AWS Weißbuch

Verfügbarkeit und mehr: Verständnis und Verbesserung der Widerstandsfähigkeit verteilter Systeme auf AWS

Verfügbarkeit und mehr: Verständnis und Verbesserung der Widerstandsfähigkeit verteilter Systeme auf AWS: AWS Weißbuch

Table of Contents

Verfügbarkeit und mehr: Die Resilienz verteilter Systeme verstehen und verbessern AWS

Datum der Veröffentlichung: 12. November 2021 (Dokumentverlauf)

Heute betreiben Unternehmen komplexe, verteilte Systeme sowohl in der Cloud als auch vor Ort. Sie möchten, dass diese Workloads widerstandsfähig sind, um ihre Kunden zu bedienen und ihre Geschäftsergebnisse zu erzielen. In diesem Whitepaper wird ein allgemeines Verständnis von Verfügbarkeit als Maß für Resilienz dargelegt, Regeln für den Aufbau hochverfügbarer Workloads festgelegt und Anleitungen zur Verbesserung der Workload-Verfügbarkeit gegeben.

Einführung

Was bedeutet es, einen hochverfügbaren Workload aufzubauen? Wie misst man die Verfügbarkeit? Was kann ich tun, um die Verfügbarkeit meines Workloads zu erhöhen? Dieses Dokument hilft Ihnen bei der Beantwortung solcher Fragen. Es ist in drei Hauptabschnitte unterteilt. Der erste Abschnitt, Verfügbarkeit verstehen, ist größtenteils theoretisch. Es schafft ein gemeinsames Verständnis der Definition von Verfügbarkeit und der Faktoren, die sich darauf auswirken. Der zweite Abschnitt, Messung der Verfügbarkeit, enthält Anleitungen zur empirischen Messung der Verfügbarkeit Ihres Workloads. Der dritte Abschnitt, Entwurf hochverfügbarer verteilter Systeme auf, AWS ist eine praktische Anwendung der im ersten Abschnitt vorgestellten Ideen. Darüber hinaus werden in diesem Dokument in diesen Abschnitten Regeln für den Aufbau stabiler Workloads aufgeführt. Dieses Dokument soll die Leitlinien und bewährten Verfahren unterstützen, die in der Säule „AWSWell-Architected Reliability“ vorgestellt werden.

In diesem Artikel werden Sie auf viel algebraische Mathematik stoßen. Die wichtigsten Erkenntnisse sind die Konzepte, die diese Mathematik unterstützt, nicht die Mathematik selbst. Nichtsdestotrotz ist es auch die Absicht dieses Papiers, eine Herausforderung darzustellen. Wenn Sie hochverfügbare Workloads betreiben, müssen Sie in der Lage sein, mathematisch nachzuweisen, dass das, was Sie erstellt haben, das erreicht, was Sie beabsichtigt haben. Selbst die besten Designs, die auf guten Absichten basieren, erzielen möglicherweise nicht immer das gewünschte Ergebnis. Das bedeutet, dass Sie Mechanismen benötigen, die die Effektivität der Lösung messen. Daher ist ein gewisses Maß an Mathematik erforderlich, um belastbare, hochverfügbare verteilte Systeme aufzubauen und zu betreiben.

Verfügbarkeit verstehen

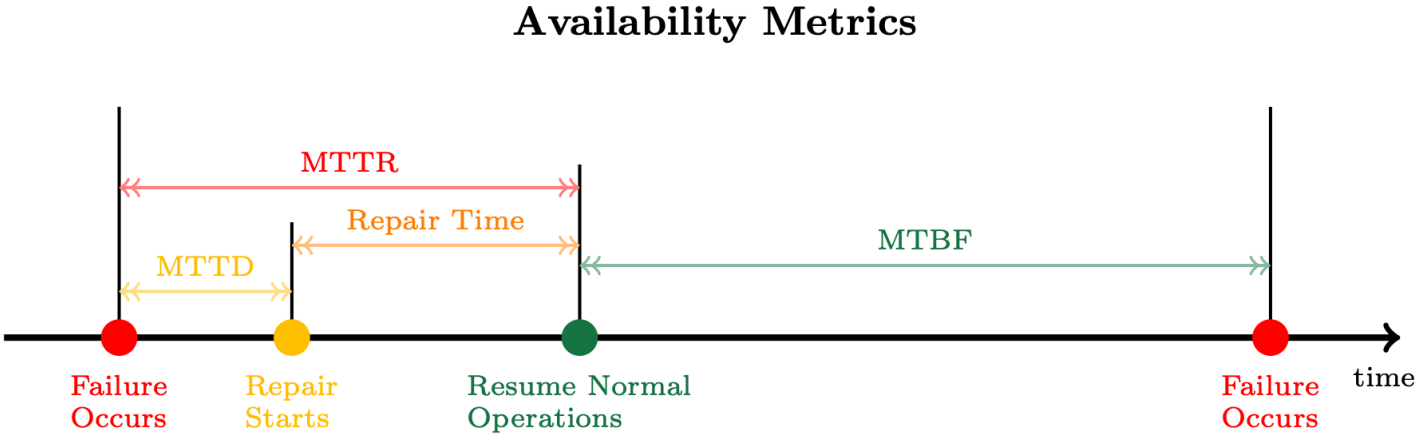
Verfügbarkeit ist eine der wichtigsten Methoden, mit denen wir die Ausfallsicherheit quantitativ messen können. Wir definieren Verfügbarkeit, A, als den Prozentsatz der Zeit, in der ein Workload zur Nutzung verfügbar ist. Es ist ein Verhältnis der erwarteten „Verfügbarkeit“ (Verfügbarkeit) zur gemessenen Gesamtzeit (der erwarteten „Betriebszeit“ plus der erwarteten „Ausfallzeit“).

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

Gleichung 1 — Verfügbarkeit

Um diese Formel besser zu verstehen, schauen wir uns an, wie Verfügbarkeit und Ausfallzeit gemessen werden können. Zunächst möchten wir wissen, wie lange der Workload fehlerfrei sein wird. Das nennen wir MTBF (Mean Time Between Failure). Dabei handelt es sich um die durchschnittliche Zeit zwischen der Aufnahme des Normalbetriebs eines Workloads und seinem nächsten Ausfall. Dann möchten wir wissen, wie lange die Wiederherstellung nach einem Ausfall dauern wird.

Wir nennen das Mean Time to Repair (or Recovery) (MTTR). Dabei handelt es sich um einen Zeitraum, in dem der Workload nicht verfügbar ist, während das ausgefallene Subsystem repariert oder wieder in Betrieb genommen wird. Ein wichtiger Zeitraum in der MTTR ist die mittlere Zeit bis zur Erkennung (MTTD), also die Zeitspanne zwischen dem Auftreten eines Fehlers und dem Beginn der Reparaturvorgänge. Das folgende Diagramm zeigt, wie all diese Metriken zusammenhängen.



Die Beziehung zwischen MTTD, MTTR und MTBF

Somit können wir die Verfügbarkeit A mithilfe von MTBF, der Zeit, zu der die Arbeitslast hoch ist, und MTTR, der Zeitpunkt, zu dem die Arbeitslast ausgefallen ist, ausdrücken.

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

Gleichung 2 — Beziehung zwischen MTBF und MTTR

Und die Wahrscheinlichkeit, dass die Arbeitslast „ausgefallen“ ist (d. h. nicht verfügbar), ist die Ausfallwahrscheinlichkeit F.

$$F = 1 - A$$

Gleichung 3 - Ausfallwahrscheinlichkeit

Zuverlässigkeit ist die Fähigkeit eines Workloads, auf Anfrage innerhalb der angegebenen Reaktionszeit das Richtige zu tun. Das ist es, was Verfügbarkeit misst. Wenn ein Workload seltener ausfällt (längere MTBF) oder eine kürzere Reparaturzeit (kürzere MTTR) hat, verbessert sich die Verfügbarkeit.

> Regel 1
>
> Weniger häufige Ausfälle (längere MTBF), kürzere Fehlererkennungszeiten (kürzere MTTD) und kürzere Reparaturzeiten (kürzere MTTR) sind die drei Faktoren, die zur Verbesserung der Verfügbarkeit in verteilten Systemen verwendet werden.

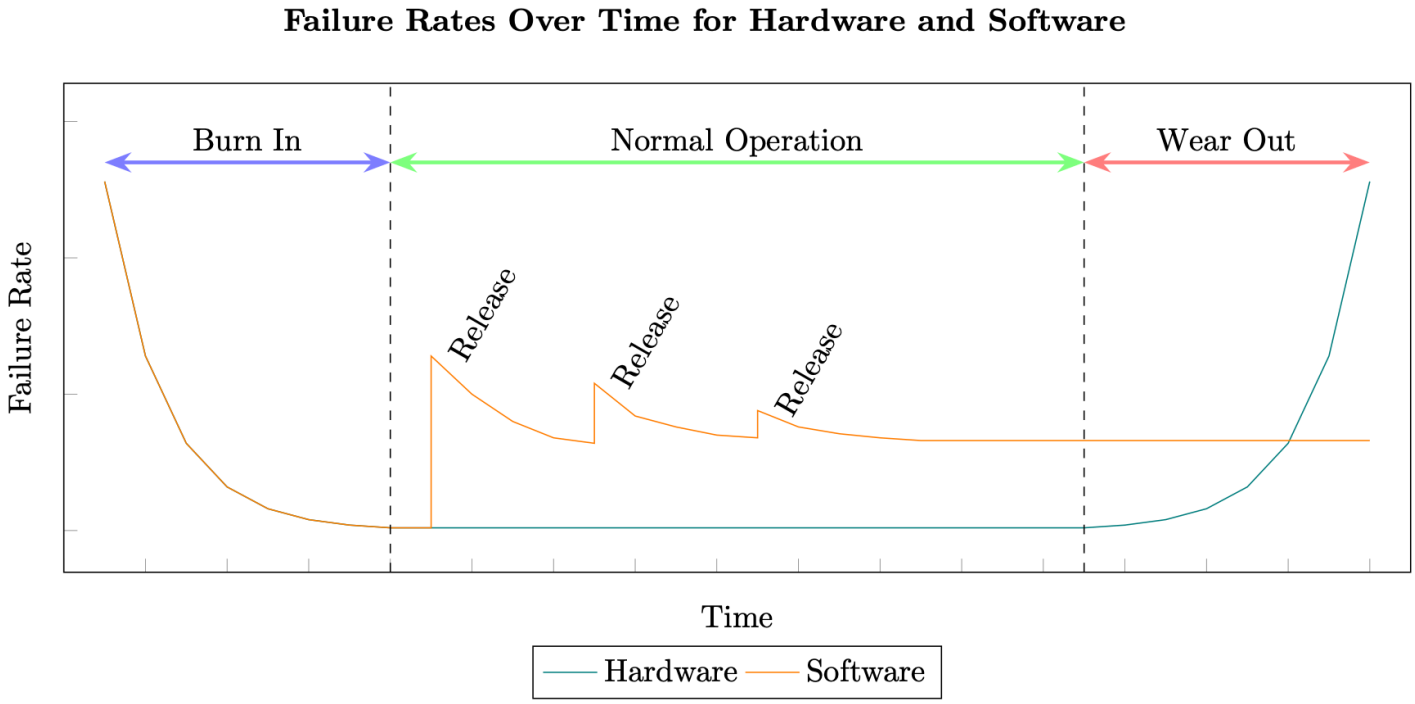
Themen

- Verfügbarkeit verteilter Systeme
- Verfügbarkeit mit Abhängigkeiten
- Verfügbarkeit mit Redundanz
- Satz von CAP
- Fehlertoleranz und Fehlerisolierung

Verfügbarkeit verteilter Systeme

Verteilte Systeme bestehen sowohl aus Software- als auch aus Hardwarekomponenten. Einige der Softwarekomponenten können selbst ein anderes verteiltes System sein. Die Verfügbarkeit sowohl der zugrunde liegenden Hardware- als auch der Softwarekomponenten wirkt sich auf die daraus resultierende Verfügbarkeit Ihres Workloads aus.

Die Berechnung der Verfügbarkeit mithilfe von MTBF und MTTR hat ihre Wurzeln in Hardwaresystemen. Verteilte Systeme scheitern jedoch aus ganz anderen Gründen als ein Teil der Hardware. Während ein Hersteller konsistent die durchschnittliche Zeit berechnen kann, bis eine Hardwarekomponente abgenutzt ist, können dieselben Tests nicht auf die Softwarekomponenten eines verteilten Systems angewendet werden. Hardware folgt in der Regel der „Badewannenkurve“ der Ausfallrate, während Software einer gestaffelten Kurve folgt, die durch zusätzliche Fehler verursacht wird, die mit jeder neuen Version auftreten (siehe Softwarezuverlässigkeit).



Hardware- und Softwareausfallraten

Darüber hinaus ändert sich die Software in verteilten Systemen in der Regel exponentiell schneller als die Hardware. Beispielsweise kann eine magnetische Standardfestplatte eine durchschnittliche jährliche Ausfallrate (AFR) von 0,93% aufweisen, was in der Praxis für eine Festplatte eine Lebensdauer von mindestens 3—5 Jahren bedeuten kann, bevor sie den Verschleißzeitraum erreicht, möglicherweise länger (siehe Backblaze Hard Drive Data and Stats, 2020). Die Festplatte

ändert sich während dieser Lebensdauer nicht wesentlich. In 3 bis 5 Jahren könnte Amazon beispielsweise mehr als 450 bis 750 Millionen Änderungen an seinen Softwaresystemen vornehmen. (Siehe Amazon Builders' Library — Automatisieren sicherer, automatischer Bereitstellungen.)

Hardware unterliegt auch dem Konzept der geplanten Obsoleszenz, d. h. sie hat eine feste Lebensdauer und muss nach einer bestimmten Zeit ausgetauscht werden. (Siehe Die große Glühbirnenverschwörung.) Software unterliegt theoretisch nicht dieser Beschränkung, sie hat keine Abnutzungszeit und kann unbegrenzt betrieben werden.

All dies bedeutet, dass dieselben Test- und Prognosemodelle, die für Hardware zur Generierung von MTBF- und MTTR-Zahlen verwendet werden, nicht für Software gelten. Seit den 1970er Jahren gab es Hunderte von Versuchen, Modelle zur Lösung dieses Problems zu entwickeln, aber sie lassen sich im Allgemeinen alle in zwei Kategorien einteilen: Vorhersagemodellierung und Schätzmodellierung. (Siehe Liste der Modelle zur Softwarezuverlässigkeit.)

Daher wird die Berechnung einer vorausschauenden MTBF und MTTR für verteilte Systeme und somit einer vorausschauenden Verfügbarkeit immer von einer Art von Vorhersage oder Prognose abgeleitet. Sie können durch prädiktive Modellierung, stochastische Simulation, historische Analysen oder strenge Tests generiert werden, aber diese Berechnungen sind keine Garantie für Verfügbarkeit oder Ausfallzeit.

Die Gründe, warum ein verteiltes System in der Vergangenheit ausgefallen ist, werden sich möglicherweise nie wiederholen. Die Gründe, warum es in future scheitert, sind wahrscheinlich unterschiedlich und möglicherweise nicht erkennbar. Die erforderlichen Wiederherstellungsmechanismen können sich auch für future Ausfälle von denen unterscheiden, die in der Vergangenheit verwendet wurden, und sie können erheblich unterschiedliche Zeiträume in Anspruch nehmen.

Außerdem handelt es sich bei MTBF und MTTR um Durchschnittswerte. Es wird eine gewisse Abweichung zwischen dem Durchschnittswert und den tatsächlich gemessenen Werten geben (die Standardabweichung, σ, misst diese Variation). Daher kann es bei Workloads im Produktionsbetrieb zu kürzeren oder längeren Zeitabständen zwischen Ausfällen und Wiederherstellungen kommen.

Davon abgesehen ist die Verfügbarkeit der Softwarekomponenten, aus denen ein verteiltes System besteht, nach wie vor wichtig. Software kann aus zahlreichen Gründen ausfallen (mehr dazu im nächsten Abschnitt) und beeinträchtigt die Verfügbarkeit der Arbeitslast. Daher sollte bei hochverfügbaren verteilten Systemen der Schwerpunkt auf die Berechnung, Messung und Verbesserung der Verfügbarkeit von Softwarekomponenten ebenso gelegt werden wie auf Hardware- und externe Softwaresubsysteme.

Regel 2

Die Verfügbarkeit der Software in Ihrem Workload ist ein wichtiger Faktor für die Gesamtverfügbarkeit Ihres Workloads und sollte ebenso berücksichtigt werden wie andere Komponenten.

Es ist wichtig zu beachten, dass MTBF und MTTR für verteilte Systeme zwar schwer vorherzusagen sind, sie aber dennoch wichtige Erkenntnisse zur Verbesserung der Verfügbarkeit bieten. Die Verringerung der Ausfallhäufigkeit (höhere MTBF) und die Verkürzung der Wiederherstellungszeit nach einem Ausfall (niedrigere MTTR) werden beide zu einer höheren empirischen Verfügbarkeit führen.

Arten von Ausfällen in verteilten Systemen

In verteilten Systemen gibt es im Allgemeinen zwei Arten von Fehlern, die sich auf die Verfügbarkeit auswirken. Sie werden liebevoll Bohrbug und Heisenbug genannt (siehe „A Conversation with Bruce Lindsay“, ACM Queue Band 2, Nr. 8 — November 2004).

Ein Bohrbug ist ein wiederholbares funktionales Softwareproblem. Bei derselben Eingabe erzeugt der Fehler immer dieselbe falsche Ausgabe (wie das deterministische Bohr-Atommodell, das solide und leicht zu erkennen ist). Diese Arten von Fehlern treten selten auf, wenn ein Workload in Betrieb genommen wird.

Ein Heisenbug ist ein vorübergehender Fehler, was bedeutet, dass er nur unter bestimmten und ungewöhnlichen Bedingungen auftritt. Diese Bedingungen beziehen sich normalerweise auf Dinge wie Hardware (z. B. ein vorübergehender Gerätefehler oder Besonderheiten der Hardwareimplementierung wie die Registergröße), Compileroptimierungen und Sprachimplementierung, Grenzbedingungen (z. B. vorübergehender Speichermangel) oder Race-Bedingungen (z. B. die Nichtverwendung einer Semaphore für Multithread-Operationen).

Heisenbugs machen den Großteil der Fehler in der Produktion aus und sind schwer zu finden, da sie schwer zu finden sind und ihr Verhalten zu ändern scheinen oder zu verschwinden scheinen, wenn man versucht, sie zu beobachten oder zu debuggen. Wenn Sie das Programm jedoch neu starten, wird der fehlgeschlagene Vorgang wahrscheinlich erfolgreich sein, da die Betriebsumgebung etwas anders ist, wodurch die Bedingungen, die den Heisenbug verursacht haben, beseitigt sind.

Daher sind die meisten Produktionsausfälle vorübergehender Natur, und wenn der Vorgang erneut versucht wird, ist es unwahrscheinlich, dass er erneut fehlschlägt. Um widerstandsfähig zu sein,

müssen verteilte Systeme fehlertolerant gegenüber Heisenbugs sein. Wie dies erreicht werden kann, werden wir im Abschnitt Erhöhung der MTBF für verteilte Systeme untersuchen.

Verfügbarkeit mit Abhängigkeiten

Im vorherigen Abschnitt haben wir erwähnt, dass Hardware, Software und möglicherweise andere verteilte Systeme allesamt Komponenten Ihres Workloads sind. Wir nennen diese Komponenten Abhängigkeiten, also die Dinge, von denen Ihr Workload abhängt, um seine Funktionalität bereitzustellen. Es gibt harte Abhängigkeiten, also Dinge, ohne die Ihr Workload nicht funktionieren kann, und weiche Abhängigkeiten, deren Nichtverfügbarkeit für einen gewissen Zeitraum unbemerkt bleiben oder toleriert werden kann. Harte Abhängigkeiten wirken sich direkt auf die Verfügbarkeit Ihres Workloads aus.

Vielleicht möchten wir versuchen, die theoretische maximale Verfügbarkeit eines Workloads zu berechnen. Dies ist das Produkt der Verfügbarkeit aller Abhängigkeiten, einschließlich der Software selbst (α_n ist die Verfügbarkeit eines einzelnen Subsystems), da jedes einzelne Subsystem betriebsbereit sein muss.

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

Gleichung 4 — Theoretische Höchstverfügbarkeit

Die in diesen Berechnungen verwendeten Verfügbarkeitszahlen beziehen sich normalerweise auf Dinge wie SLAs oder Service Level Objectives (SLOs). SLAs definieren das erwartete Serviceniveau, das Kunden erhalten, die Kennzahlen, anhand derer der Service gemessen wird, und Abhilfemaßnahmen oder Strafen (in der Regel monetär) für den Fall, dass das Serviceniveau nicht erreicht wird.

Anhand der obigen Formel können wir schlussfolgern, dass ein Workload rein mathematisch gesehen nicht mehr verfügbar sein kann als jede seiner Abhängigkeiten. In der Realität sehen wir jedoch normalerweise, dass dies nicht der Fall ist. Ein Workload, der aus zwei oder drei Abhängigkeiten mit SLAs für eine Verfügbarkeit von 99,99% erstellt wurde, kann selbst immer noch eine Verfügbarkeit von 99,99% oder höher erreichen.

Dies liegt daran, dass es sich bei diesen Verfügbarkeitszahlen, wie wir im vorherigen Abschnitt dargelegt haben, um Schätzungen handelt. Sie schätzen oder prognostizieren, wie oft ein Fehler auftritt und wie schnell er repariert werden kann. Sie sind keine Garantie für Ausfallzeiten. Abhängigkeiten überschreiten häufig ihre angegebene Verfügbarkeit (SLA oder SLO).

Abhängigkeiten können auch höhere interne Verfügbarkeitsziele haben, auf deren Grundlage sie die Leistung anstreben, als die in öffentlichen SLAs angegebenen Zahlen. Dies bietet ein gewisses Maß an Risikominderung bei der Einhaltung von SLAs, wenn etwas Unbekanntes oder Unbekanntes passiert.

Schließlich kann es sein, dass Ihr Workload Abhängigkeiten aufweist, deren SLAs nicht bekannt sind oder die kein SLA oder SLO bieten. Beispielsweise ist weltweites Internet-Routing eine häufige Abhängigkeit für viele Workloads, aber es ist schwierig zu wissen, welche Internetdienstanbieter Ihr globaler Traffic nutzt, ob sie SLAs haben und wie konsistent diese zwischen den Anbietern sind.

All dies zeigt uns, dass die Berechnung einer maximalen theoretischen Verfügbarkeit wahrscheinlich nur zu einer groben Berechnung der Größenordnung führt, aber für sich genommen wahrscheinlich nicht genau ist oder aussagekräftige Erkenntnisse liefert. Die Mathematik zeigt uns, dass die Wahrscheinlichkeit eines Fehlers insgesamt sinkt, je weniger Dinge Ihre Arbeitslast benötigt. Je weniger Zahlen kleiner als eins miteinander multipliziert werden, desto größer ist das Ergebnis.

> **Regel 3**
>
> Die Reduzierung von Abhängigkeiten kann sich positiv auf die Verfügbarkeit auswirken.

Die Mathematik hilft auch bei der Auswahl von Abhängigkeiten. Der Auswahlprozess wirkt sich darauf aus, wie Sie Ihre Arbeitslast entwerfen, wie Sie Redundanz in Abhängigkeiten nutzen, um deren Verfügbarkeit zu verbessern, und ob Sie diese Abhängigkeiten als weich oder hart betrachten. Abhängigkeiten, die sich auf Ihre Arbeitslast auswirken können, sollten sorgfältig ausgewählt werden. Die nächste Regel enthält Hinweise dazu, wie Sie dies tun können.

> **Regel 4**
>
> Wählen Sie im Allgemeinen Abhängigkeiten aus, deren Verfügbarkeitsziele den Zielen Ihres Workloads entsprechen oder diese übertreffen.

Verfügbarkeit mit Redundanz

Wenn ein Workload mehrere, unabhängige und redundante Subsysteme nutzt, kann ein höheres Maß an theoretischer Verfügbarkeit erreicht werden, als wenn ein einzelnes Subsystem verwendet würde. Stellen Sie sich zum Beispiel eine Arbeitslast vor, die aus zwei identischen Subsystemen

besteht. Es kann vollständig betriebsbereit sein, wenn entweder das erste Teilsystem oder das zweite Teilsystem betriebsbereit ist. Damit das gesamte System ausgefallen ist, müssen beide Teilsysteme gleichzeitig ausgefallen sein.

Wenn die Ausfallwahrscheinlichkeit eines Teilsystems 1 − α ist, dann ist die Wahrscheinlichkeit, dass zwei redundante Teilsysteme gleichzeitig ausfallen, das Produkt der Ausfallwahrscheinlichkeit jedes Teilsystems, F = (1− α_1) × (1− α). $_2$ Für eine Arbeitslast mit zwei redundanten Subsystemen ergibt sich unter Verwendung von Gleichung (3) eine Verfügbarkeit, die wie folgt definiert ist:

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

Gleichung 5

Für zwei Subsysteme mit einer Verfügbarkeit von 99% beträgt die Wahrscheinlichkeit, dass eines ausfällt, also 1% und die Wahrscheinlichkeit, dass beide ausfallen, beträgt (1− 99%) × (1− 99%) = 0,01%. Das macht die Verfügbarkeit bei Verwendung von zwei redundanten Subsystemen zu 99,99%.

Dies kann generalisiert werden, um auch zusätzliche redundante Ersatzteile einzubeziehen. In Gleichung (5) wurde nur von einem einzigen Ersatzgerät ausgegangen, aber ein Workload kann aus zwei, drei oder mehr Ersatzteilen bestehen, sodass er den gleichzeitigen Verlust mehrerer Subsysteme überstehen kann, ohne die Verfügbarkeit zu beeinträchtigen. Wenn eine Arbeitslast aus drei Subsystemen besteht und zwei Ersatzsysteme sind, beträgt die Wahrscheinlichkeit, dass alle drei Subsysteme gleichzeitig ausfallen, (1− α) × (1− α) × (1− α) oder (1− α) 3. Im Allgemeinen schlägt ein Workload mit S-Ersatzteilen nur fehl, wenn s + 1-Subsysteme ausfallen.

Bei einer Arbeitslast mit n Subsystemen und s Spares ist f die Anzahl der Ausfallursachen oder die Art und Weise, wie s + 1-Subsysteme von n ausfallen können.

Das ist quasi der Binomialsatz, die kombinatorische Mathematik, bei der k Elemente aus einer Menge von n ausgewählt werden, oder „n wähle k“. In diesem Fall ist k s + 1.

$$k = s + 1$$

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

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

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

Gleichung 6

Wir können dann eine allgemeine Näherung der Verfügbarkeit erstellen, die die Anzahl der Ausfallursachen und die Einsparung berücksichtigt. (Um zu verstehen, warum das ungefähr so ist, siehe Anhang 2 von Highleyman et al. Die Verfügbarkeitsbarriere durchbrechen.)

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

Gleichung 7

Sparing kann auf jede Abhängigkeit angewendet werden, die Ressourcen bereitstellt, die unabhängig voneinander ausfallen. Beispiele hierfür AWS-Regionen sind Amazon EC2 EC2-Instances in verschiedenen AZs oder Amazon S3 S3-Buckets in verschiedenen. Die Verwendung von Ersatzteilen trägt dazu bei, dass diese Abhängigkeit eine höhere Gesamtverfügbarkeit erreicht, um die Verfügbarkeitsziele des Workloads zu erreichen.

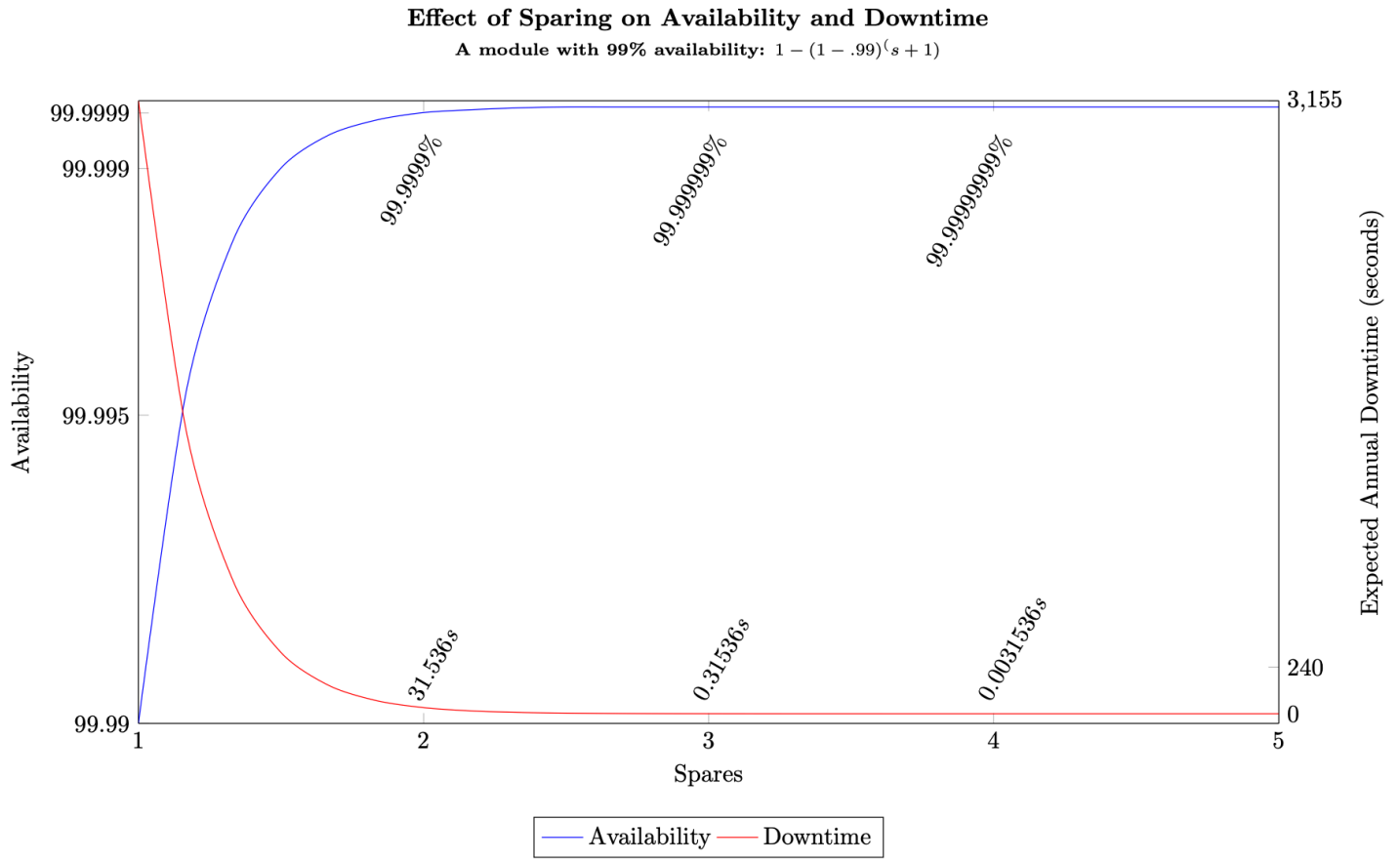
Regel 5

Verwenden Sie Sparing, um die Verfügbarkeit von Abhängigkeiten in einem Workload zu erhöhen.

Sparing ist jedoch mit Kosten verbunden. Jedes weitere Ersatzteil kostet genauso viel wie das Originalmodul, was die Kosten zumindest linear ansteigen lässt. Der Aufbau einer Workload, die Ersatzteile verwenden kann, erhöht auch deren Komplexität. Es muss wissen, wie man Fehler in Abhängigkeit erkennt, die Arbeit davon abwälzt und auf eine gesunde Ressource umverteilt und die Gesamtkapazität der Arbeitslast verwaltet.

Redundanz ist ein Optimierungsproblem. Zu wenige Ersatzteile, und der Workload kann häufiger ausfallen als gewünscht, zu viele Ersatzteile und der Betrieb des Workloads kostet zu viel. Es gibt einen bestimmten Schwellenwert, ab dem das Hinzufügen weiterer Ersatzteile mehr kostet als die zusätzliche Verfügbarkeit, die sie erreichen.

Unter Verwendung unserer Formel für allgemeine Verfügbarkeit mit Ersatzteilen, Gleichung (7), für ein Subsystem mit einer Verfügbarkeit von 99,5% beträgt die Verfügbarkeit der Arbeitslast bei zwei Ersatzteilen $A \approx 1 - (1)\ (1-.995)^{3} = 99{,}9999875\%$ (ca. 3,94 Sekunden Ausfallzeit pro Jahr), und bei 10 Ersatzteilen erhalten wir $A \approx 1 - (1)\ (1-.995)^{11} = 25{,}5\ 9's$ (die ungefähre Ausfallzeit) wäre $1{,}26252 \times 10^{-15}$ m s pro Jahr, effektiv 0). Beim Vergleich dieser beiden Workloads haben wir die Kosten für das Sparen um das Fünffache erhöht, sodass wir die Ausfallzeiten pro Jahr um vier Sekunden reduzieren konnten. Bei den meisten Workloads wäre der Anstieg der Kosten aufgrund dieser Erhöhung der Verfügbarkeit ungerechtfertigt. Die folgende Abbildung zeigt diesen Zusammenhang.



Sinkende Renditen aufgrund verstärkter Sparsamkeit

Bei drei Ersatzteilen und mehr führt dies zu Sekundenbruchteilen der erwarteten Ausfallzeit pro Jahr, was bedeutet, dass Sie nach diesem Zeitpunkt den Bereich mit sinkenden Renditen erreichen. Möglicherweise besteht der Drang, „einfach mehr hinzuzufügen“, um eine höhere Verfügbarkeit zu erreichen, aber in Wirklichkeit verschwindet der Kostenvorteil sehr schnell. Die Verwendung von mehr als drei Ersatzteilen bringt keinen wesentlichen Nutzen. Ein spürbarer Gewinn für fast alle Workloads, wenn das Subsystem selbst eine Verfügbarkeit von mindestens 99% aufweist.

> **Regel 6**
>
> Es gibt eine Obergrenze für die Kosteneffizienz von Sparsamkeit. Verwenden Sie die wenigsten Ersatzteile, die erforderlich sind, um die erforderliche Verfügbarkeit zu erreichen.

Bei der Auswahl der richtigen Anzahl von Ersatzteilen sollten Sie die Fehlereinheit berücksichtigen. Schauen wir uns zum Beispiel einen Workload an, für den 10 EC2-Instances erforderlich sind, um Spitzenkapazität zu bewältigen, und die in einer einzigen AZ bereitgestellt werden.

Da AZs als Grenzen zur Fehlerisolierung konzipiert sind, handelt es sich bei der Ausfalleinheit nicht nur um eine einzelne EC2-Instance, da eine gesamte Anzahl von EC2-Instances zusammen ausfallen kann. In diesem Fall sollten Sie die Redundanz durch eine weitere AZ erhöhen und 10 zusätzliche EC2-Instances bereitstellen, um die Last im Fall eines AZ-Ausfalls zu bewältigen, also insgesamt 20 EC2-Instances (entsprechend dem Muster der statischen Stabilität).

Dies scheint zwar 10 Ersatz-EC2-Instances zu sein, aber in Wirklichkeit handelt es sich nur um eine einzige Ersatz-AZ, sodass wir den Punkt sinkender Renditen nicht überschritten haben. Sie können jedoch noch kosteneffizienter arbeiten und gleichzeitig Ihre Verfügbarkeit erhöhen, indem Sie drei AZs verwenden und fünf EC2-Instances pro AZ bereitstellen.

Auf diese Weise steht eine Ersatz-AZ mit insgesamt 15 EC2-Instances (gegenüber zwei AZs mit 20 Instances) zur Verfügung, sodass immer noch die insgesamt erforderlichen 10 Instances zur Verfügung stehen, um die Spitzenkapazität während eines Ereignisses, das sich auf eine einzelne AZ auswirkt, zu bedienen. Daher sollten Sie Sparing einbauen, um über alle vom Workload verwendeten Grenzen der Fehlerisolierung (Instanz, Zelle, AZ und Region) hinweg fehlertolerant zu sein.

Satz von CAP

Eine andere Art, wie wir über Verfügbarkeit nachdenken könnten, bezieht sich auf das CAP-Theorem. Der Satz besagt, dass ein verteiltes System, das aus mehreren Knoten besteht, die Daten speichern, nicht gleichzeitig mehr als zwei der folgenden drei Garantien bieten kann:

- Konsistenz: Jede Leseanforderung erhält den letzten Schreibvorgang oder einen Fehler, wenn die Konsistenz nicht garantiert werden kann.
- Verfügbarkeit: Jede Anfrage erhält eine fehlerfreie Antwort, auch wenn Knoten ausgefallen oder nicht verfügbar sind.
- Partitionstoleranz: Das System arbeitet trotz des Verlusts einer beliebigen Anzahl von Nachrichten zwischen Knoten weiter.

(Weitere Einzelheiten finden Sie in Seth Gilbert und Nancy Lynch, Brewers Vermutung und die Machbarkeit konsistenter, verfügbarer, partitionstoleranter Webdienste, ACM SIGACT News, Band 33, Ausgabe 2 (2002), S. 51—59.)

Die meisten verteilten Systeme müssen Netzwerkausfälle tolerieren, weshalb Netzwerkpartitionierung zulässig sein muss. Das bedeutet, dass diese Workloads bei einer Netzwerkpartition zwischen Konsistenz und Verfügbarkeit wählen müssen. Wenn sich der Workload für Verfügbarkeit entscheidet, gibt er immer eine Antwort zurück, allerdings mit potenziell inkonsistenten Daten.

Wenn er sich für Konsistenz entscheidet, würde er während einer Netzwerkpartition einen Fehler zurückgeben, da der Workload nicht sicher sein kann, ob die Daten konsistent sind.

Für Workloads, deren Ziel es ist, ein höheres Maß an Verfügbarkeit zu gewährleisten, könnten sie Verfügbarkeit und Partitionstoleranz (AP) wählen, um zu verhindern, dass während einer Netzwerkpartition Fehler (Nichtverfügbarkeit) erneut auftreten. Dies führt dazu, dass ein lockereres Konsistenzmodell erforderlich ist, wie z. B. letztendliche Konsistenz oder monotone Konsistenz.

Fehlertoleranz und Fehlerisolierung

Dies sind zwei wichtige Konzepte, wenn wir über Verfügbarkeit nachdenken. Fehlertoleranz ist die Fähigkeit, dem Ausfall eines Subsystems standzuhalten und die Verfügbarkeit aufrechtzuerhalten (das Richtige innerhalb einer festgelegten SLA zu tun). Um Fehlertoleranz zu implementieren, verwenden Workloads Ersatzsubsysteme (oder redundante). Wenn eines der Subsysteme in einer redundanten Gruppe ausfällt, nimmt ein anderes seine Arbeit wieder auf, normalerweise fast nahtlos. In diesem Fall handelt es sich bei Ersatzteilen um echte Reservekapazitäten. Sie stehen zur Verfügung, um 100% der Arbeit des ausgefallenen Subsystems zu übernehmen. Bei echten Ersatzteilen sind mehrere Ausfälle von Subsystemen erforderlich, um sich negativ auf die Arbeitslast auszuwirken.

Durch die Fehlerisolierung wird das Ausmaß der Auswirkungen minimiert, wenn ein Fehler auftritt. Dies wird in der Regel durch Modularisierung implementiert. Workloads werden in kleine Subsysteme aufgeteilt, die unabhängig voneinander ausfallen und isoliert repariert werden können. Der Ausfall eines Moduls überträgt sich nicht über das Modul hinaus. Diese Idee erstreckt sich sowohl vertikal über unterschiedliche Funktionen innerhalb eines Workloads als auch horizontal über mehrere Subsysteme, die dieselbe Funktionalität bieten. Diese Module dienen als Fehlercontainer, die den Umfang der Auswirkungen während eines Ereignisses einschränken.

Die architektonischen Muster der Steuerungsebenen, Datenebenen und der statischen Stabilität unterstützen direkt die Implementierung von Fehlertoleranz und Fehlerisolierung. Der Artikel Statische Stabilität mithilfe von Availability Zones in der Amazon Builders' Library bietet gute Definitionen für diese Begriffe und erklärt, wie sie sich auf den Aufbau robuster, hochverfügbarer Workloads anwenden lassen. In diesem Whitepaper werden diese Muster im Abschnitt Entwerfen hochverfügbarer verteilter Systeme verwendet. Außerdem fassen wir ihre Definitionen hier zusammen. AWS

- Kontrollebene — Der Teil der Arbeitslast, der mit der Durchführung von Änderungen verbunden ist: Hinzufügen von Ressourcen, Löschen von Ressourcen, Ändern von Ressourcen und

Weitergabe dieser Änderungen an die Stellen, an denen sie benötigt werden. Steuerungsebenen sind in der Regel komplexer und haben mehr bewegliche Teile als Datenebenen. Daher ist die Wahrscheinlichkeit, dass sie ausfallen und weniger verfügbar sind, statistisch gesehen höher.

- Datenebene — Der Teil der Arbeitslast, der die day-to-day Geschäftsfunktionen bereitstellt. Datenebenen sind in der Regel einfacher und können mit höheren Datenvolumen betrieben werden als Steuerungsebenen, was zu höheren Verfügbarkeiten führt.
- Statische Stabilität — Die Fähigkeit eines Workloads, trotz eingeschränkter Abhängigkeiten den korrekten Betrieb fortzusetzen. Eine Implementierungsmethode besteht darin, Abhängigkeiten der Steuerungsebene von den Datenebenen zu entfernen. Eine andere Methode besteht darin, Workload-Abhängigkeiten lose miteinander zu koppeln. Möglicherweise sieht der Workload keine aktualisierten Informationen (wie neue Dinge, gelöschte oder geänderte Dinge), die seine Abhängigkeit hätte liefern sollen. Alles, was es getan hat, bevor die Abhängigkeit beeinträchtigt wurde, funktioniert jedoch weiterhin.

Wenn wir über eine Beeinträchtigung der Arbeitsbelastung nachdenken, gibt es zwei grundlegende Ansätze, die wir für die Wiederherstellung in Betracht ziehen können. Die erste Methode besteht darin, auf diese Beeinträchtigung zu reagieren, nachdem sie eingetreten ist, indem AWS Auto Scaling möglicherweise neue Kapazitäten hinzugefügt werden. Die zweite Methode besteht darin, sich auf diese Beeinträchtigungen vorzubereiten, bevor sie auftreten, etwa indem die Infrastruktur eines Workloads überdimensioniert wird, sodass dieser weiterhin ordnungsgemäß funktionieren kann, ohne dass zusätzliche Ressourcen benötigt werden.

Ein statisch stabiles System verwendet den letztgenannten Ansatz. Es stellt vorab Kapazitätsreserven bereit, die bei einem Ausfall zur Verfügung stehen. Mit dieser Methode wird vermieden, dass im Wiederherstellungspfad des Workloads eine Abhängigkeit von einer Steuerungsebene entsteht, um neue Kapazitäten für die Wiederherstellung nach einem Ausfall bereitzustellen. Darüber hinaus nimmt die Bereitstellung neuer Kapazitäten für verschiedene Ressourcen Zeit in Anspruch. Während Sie auf neue Kapazitäten warten, kann es sein, dass Ihr Workload durch die bestehende Nachfrage überlastet wird und sich weiter verschlechtert, was zu einem „Brown-Out“ oder einem vollständigen Verfügbarkeitsverlust führt. Sie sollten jedoch auch die Kostenauswirkungen berücksichtigen, die sich aus der Nutzung vorab bereitgestellter Kapazität im Vergleich zu Ihren Verfügbarkeitszielen ergeben.

Statische Stabilität gibt die nächsten beiden Regeln für Hochverfügbarkeits-Workloads vor.

Regel 7

Gehen Sie in Ihrer Datenebene keine Abhängigkeiten von den Steuerungsebenen ein, insbesondere nicht bei der Wiederherstellung.

Regel 8

Koppeln Sie Abhängigkeiten nach Möglichkeit lose miteinander, sodass Ihr Workload trotz Beeinträchtigung der Abhängigkeit korrekt funktionieren kann.

Messung der Verfügbarkeit

Wie wir bereits gesehen haben, ist die Erstellung eines zukunftsorientierten Verfügbarkeitsmodells für ein verteiltes System schwierig und bietet möglicherweise nicht die gewünschten Erkenntnisse. Was mehr Nutzen bringen kann, ist die Entwicklung einheitlicher Methoden zur Messung der Verfügbarkeit Ihres Workloads.

Die Definition von Verfügbarkeit als Verfügbarkeit und Ausfallzeit stellt einen Ausfall als binäre Option dar, entweder ist der Workload gestiegen oder nicht.

Dies ist jedoch selten der Fall. Ausfälle haben gewisse Auswirkungen und treten häufig auf einen Teil der Arbeitslast auf. Sie betreffen einen Prozentsatz der Benutzer oder Anfragen, einen Prozentsatz der Standorte oder ein Perzentil der Latenz. Dies sind alles Teilfehlermodi.

Und obwohl MTTR und MTBF nützlich sind, um zu verstehen, was die resultierende Verfügbarkeit eines Systems beeinflusst und wie es verbessert werden kann, dienen sie nicht als empirisches Maß für die Verfügbarkeit. Darüber hinaus bestehen Workloads aus vielen Komponenten. Ein Workload wie ein Zahlungsabwicklungssystem besteht beispielsweise aus vielen Anwendungsprogrammierschnittstellen (APIs) und Subsystemen. Also, wenn wir eine Frage stellen wollen wie: „Wie ist die Verfügbarkeit des gesamten Workloads?“ , es ist eigentlich eine komplexe und nuancierte Frage.

In diesem Abschnitt werden wir drei Möglichkeiten untersuchen, wie die Verfügbarkeit empirisch gemessen werden kann: Erfolgsrate serverseitiger Anfragen, Erfolgsquote clientseitiger Anfragen und jährliche Ausfallzeiten.

Erfolgsquote serverseitiger und clientseitiger Anfragen

Die ersten beiden Methoden sind sich sehr ähnlich und unterscheiden sich nur in der Sichtweise, in der die Messung durchgeführt wird. Serverseitige Metriken können anhand der Instrumentierung im Service erfasst werden. Sie sind jedoch nicht vollständig. Wenn Kunden den Service nicht erreichen können, können Sie diese Kennzahlen nicht erfassen. Um das Kundenerlebnis zu verstehen, ist es einfacher, kundenseitige Metriken zu sammeln, anstatt sich auf Telemetrie von Kunden zu verlassen, wenn es um fehlgeschlagene Anfragen geht, indem Sie den Kundenverkehr mit Canaries simulieren, einer Software, die Ihre Dienste regelmäßig überprüft und Kennzahlen aufzeichnet.

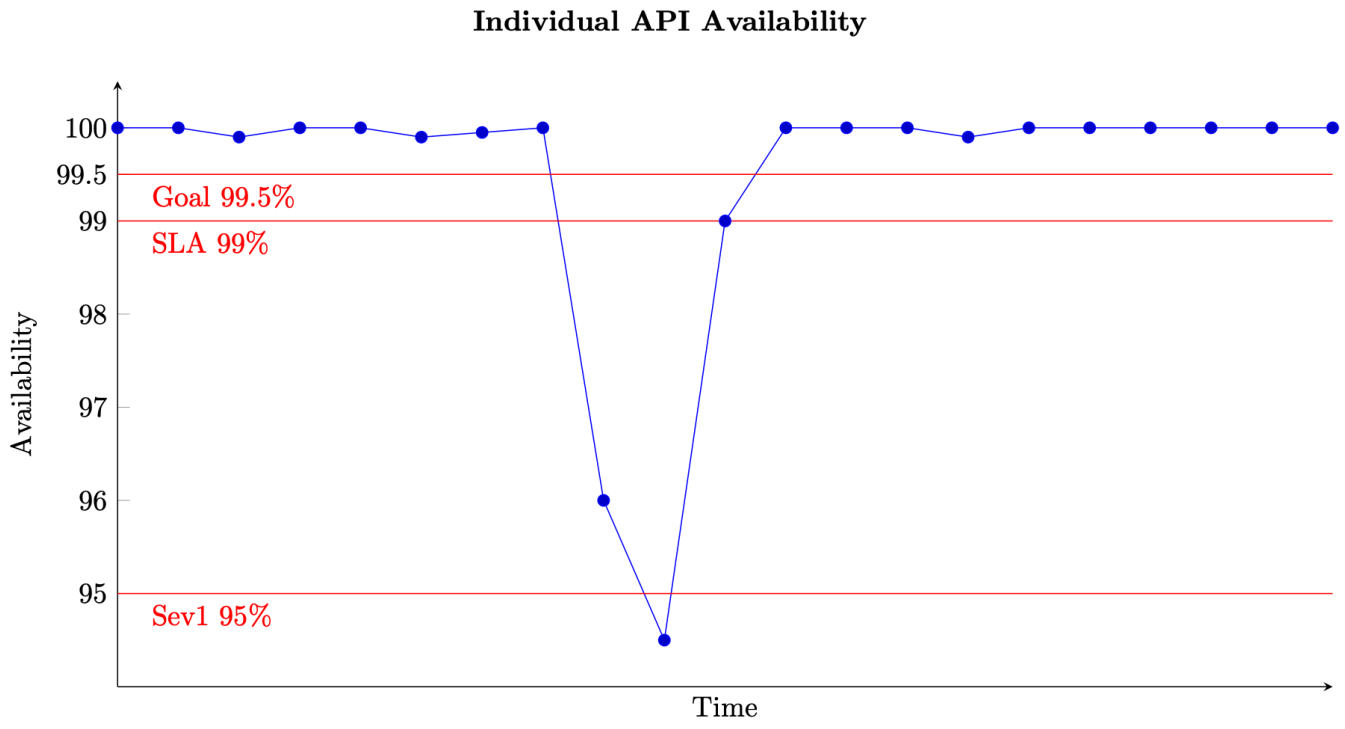
Diese beiden Methoden berechnen die Verfügbarkeit als den Bruchteil der gesamten gültigen Arbeitseinheiten, die der Dienst empfängt, und der Arbeitseinheiten, die er erfolgreich verarbeitet hat

(dabei werden ungültige Arbeitseinheiten ignoriert, z. B. eine HTTP-Anfrage, die zu einem 404-Fehler führt).

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

Gleichung 8

Bei einem auf Anfrage basierenden Dienst ist die Arbeitseinheit die Anfrage, wie bei einer HTTP-Anfrage. Bei ereignisbasierten oder aufgabenbasierten Diensten handelt es sich bei den Arbeitseinheiten um Ereignisse oder Aufgaben, z. B. die Verarbeitung einer Nachricht aus einer Warteschlange. Dieses Maß für die Verfügbarkeit ist in kurzen Zeitintervallen, z. B. in Zeitfenstern von einer Minute oder fünf Minuten, aussagekräftig. Es eignet sich auch am besten für eine granulare Perspektive, z. B. auf API-Ebene für einen anforderungsbasierten Dienst. Die folgende Abbildung zeigt, wie die Verfügbarkeit im Laufe der Zeit aussehen könnte, wenn sie auf diese Weise berechnet wird. Jeder Datenpunkt in der Grafik wird mithilfe von Gleichung (8) über ein Fünf-Minuten-Fenster berechnet (Sie können andere Zeitdimensionen wie Intervalle von einer Minute oder zehn Minuten wählen). Datenpunkt 10 zeigt beispielsweise eine Verfügbarkeit von 94,5%. Das bedeutet, dass in den Minuten t+45 bis t+50, wenn der Dienst 1.000 Anfragen erhielt, nur 945 davon erfolgreich bearbeitet wurden.



Beispiel für die Messung der Verfügbarkeit im Zeitverlauf für eine einzelne API

Die Grafik zeigt auch das Verfügbarkeitsziel der API, nämlich eine Verfügbarkeit von 99,5%, das Service Level Agreement (SLA), das sie den Kunden anbietet, eine Verfügbarkeit von 99% und den Schwellenwert für einen Alarm mit hohem Schweregrad von 95% Ohne den Kontext dieser verschiedenen Schwellenwerte bietet ein Verfügbarkeitsdiagramm möglicherweise keinen aussagekräftigen Einblick in die Funktionsweise Ihres Dienstes.

Wir möchten auch in der Lage sein, die Verfügbarkeit eines größeren Subsystems, wie einer Steuerungsebene, oder eines gesamten Dienstes nachzuverfolgen und zu beschreiben. Eine Möglichkeit, dies zu tun, besteht darin, den Durchschnitt jedes Fünf-Minuten-Datenpunkts für jedes Subsystem zu ermitteln. Das Diagramm sieht ähnlich aus wie das vorherige, ist aber repräsentativ für einen größeren Satz von Eingaben. Außerdem wird allen Subsystemen, aus denen Ihr Service besteht, das gleiche Gewicht beigemessen. Ein alternativer Ansatz könnte darin bestehen, alle eingegangenen und erfolgreich verarbeiteten Anfragen von allen APIs im Service zusammenzufassen, um die Verfügbarkeit in Intervallen von fünf Minuten zu berechnen.

Diese letztere Methode kann jedoch eine einzelne API verbergen, die einen niedrigen Durchsatz und eine schlechte Verfügbarkeit aufweist. Stellen Sie sich als einfaches Beispiel einen Dienst mit zwei APIs vor.

Die erste API empfängt innerhalb von fünf Minuten 1.000.000 Anfragen und verarbeitet 999.000 davon erfolgreich, was einer Verfügbarkeit von 99,9% entspricht. Die zweite API empfängt 100 Anfragen in demselben Fünf-Minuten-Fenster und verarbeitet nur 50 davon erfolgreich, was einer Verfügbarkeit von 50% entspricht.

Wenn wir die Anfragen von jeder API zusammenfassen, gibt es insgesamt 1.000.100 gültige Anfragen, von denen 999.050 erfolgreich verarbeitet wurden, was einer Verfügbarkeit des Dienstes insgesamt von 99,895% entspricht. Wenn wir jedoch den Durchschnitt der Verfügbarkeiten der beiden APIs berechnen, die erstere Methode, erhalten wir eine resultierende Verfügbarkeit von 74,95%, was möglicherweise aussagekräftiger für die tatsächliche Erfahrung ist.

Keiner der beiden Ansätze ist falsch, zeigt aber, wie wichtig es ist, zu verstehen, was Verfügbarkeitsmetriken Ihnen sagen. Sie könnten sich dafür entscheiden, Anfragen für alle Subsysteme zusammenzufassen, wenn Ihr Workload in allen Subsystemen ein ähnliches Anforderungsvolumen erhält. Dieser Ansatz konzentriert sich auf die „Anfrage“ und ihren Erfolg als Maß für die Verfügbarkeit und das Kundenerlebnis. Alternativ können Sie den Durchschnitt der Subsystemverfügbarkeiten festlegen, um deren Wichtigkeit trotz unterschiedlicher

Anforderungsmengen gleichmäßig darzustellen. Dieser Ansatz konzentriert sich auf das Subsystem und die Fähigkeit jedes einzelnen, stellvertretend für das Kundenerlebnis zu stehen.

Jährliche Ausfallzeiten

Der dritte Ansatz ist die Berechnung der jährlichen Ausfallzeiten. Diese Form der Verfügbarkeitsmetrik eignet sich eher für die Festlegung und Überprüfung längerfristiger Ziele. Dazu muss definiert werden, was Ausfallzeiten für Ihre Arbeitslast bedeuten. Anschließend können Sie die Verfügbarkeit anhand der Anzahl der Minuten messen, in denen sich der Workload nicht in einem „Ausfall“ befand, im Verhältnis zur Gesamtzahl der Minuten im angegebenen Zeitraum.

Bei einigen Workloads kann Ausfallzeit möglicherweise so definiert werden, dass die Verfügbarkeit einer einzelnen API oder Workload-Funktion für ein Intervall von einer Minute oder fünf Minuten unter 95% fällt (was in der vorherigen Verfügbarkeitsgrafik der Fall war). Sie können Ausfallzeiten auch nur in Betracht ziehen, da sie für einen Teil der kritischen Datenebenenoperationen gelten. Beispielsweise gilt das Service Level Agreement von Amazon Messaging (SQS, SNS) für die SQS-Verfügbarkeit für die SQS-API zum Senden, Empfangen und Löschen.

Größere, komplexere Workloads müssen möglicherweise systemweite Verfügbarkeitsmetriken definieren. Für eine große E-Commerce-Website kann eine systemweite Kennzahl so etwas wie die Bestellrate von Kunden sein. Hier kann ein Rückgang der Bestellungen um 10% oder mehr im Vergleich zur prognostizierten Menge innerhalb eines Fünf-Minuten-Fensters zu Ausfallzeiten führen.

Bei beiden Ansätzen können Sie dann alle Ausfallzeiten summieren, um eine jährliche Verfügbarkeit zu berechnen. Wenn es beispielsweise in einem Kalenderjahr 27 Ausfallzeiten von jeweils fünf Minuten gegeben hätte, was definiert ist, dass die Verfügbarkeit einer API auf Datenebene unter 95% fiel, betrug die Gesamtausfallzeit 135 Minuten (einige Fünf-Minuten-Perioden könnten aufeinanderfolgend gewesen sein, andere isoliert), was einer jährlichen Verfügbarkeit von 99,97% entspricht.

Diese zusätzliche Methode zur Messung der Verfügbarkeit kann Daten und Erkenntnisse liefern, die in den clientseitigen und serverseitigen Metriken fehlen. Stellen Sie sich zum Beispiel eine Arbeitslast vor, die beeinträchtigt ist und bei der die Fehlerquoten deutlich erhöht sind. Kunden mit diesem Workload könnten ganz aufhören, ihre Dienste anzurufen. Vielleicht haben sie einen Schutzschalter aktiviert oder ihren Notfallwiederherstellungsplan befolgt, um den Dienst in einer anderen Region zu nutzen. Wenn wir nur fehlgeschlagene Antworten messen würden, kann die Verfügbarkeit des Workloads während der Beeinträchtigung sogar steigen, aber nicht, weil sich die Beeinträchtigung bessert oder verschwindet, sondern weil die Kunden ihn einfach nicht mehr nutzen.

Latency

Schließlich ist es auch wichtig, die Latenz der Verarbeitung von Arbeitseinheiten innerhalb Ihres Workloads zu messen. Ein Teil der Verfügbarkeitsdefinition besteht darin, die Arbeit innerhalb einer festgelegten SLA zu erledigen. Wenn die Rückgabe einer Antwort länger dauert als das Client-Timeout, geht der Client davon aus, dass die Anfrage fehlgeschlagen ist und der Workload nicht verfügbar ist. Auf der Serverseite scheint die Anfrage jedoch erfolgreich verarbeitet worden zu sein.

Die Messung der Latenz bietet eine weitere Möglichkeit, die Verfügbarkeit zu bewerten. Die Verwendung von Perzentilen und dem getrimmten Mittelwert ist eine gute Statistik für diese Messung. Sie werden üblicherweise im 50. Perzentil (P50 und TM50) und 99. Perzentil (P99 und TM99) gemessen. Die Latenz sollte anhand von Kanariendaten gemessen werden, um das Kundenerlebnis abzubilden, sowie anhand serverseitiger Metriken. Immer wenn die durchschnittliche Latenz eines bestimmten Perzentils, wie P99 oder TM99,9, über einem Ziel-SLA liegt, können Sie diese Ausfallzeit berücksichtigen, die zu Ihrer jährlichen Ausfallzeitberechnung beiträgt.

Entwerfen hochverfügbarer verteilter Systeme auf AWS

In den vorherigen Abschnitten ging es hauptsächlich um die theoretische Verfügbarkeit von Workloads und darum, was damit erreicht werden kann. Dies sind wichtige Konzepte, die Sie beim Aufbau verteilter Systeme berücksichtigen sollten. Sie helfen Ihnen bei der Auswahl von Abhängigkeiten und bei der Implementierung von Redundanz.

Wir haben uns auch mit dem Verhältnis von MTTDMTTR, und MTBF zur Verfügbarkeit befasst. In diesem Abschnitt werden praktische Anleitungen vorgestellt, die auf der vorherigen Theorie basieren. Kurz gesagt, der technische Arbeitsaufwand für Hochverfügbarkeit zielt darauf ab, die zu erhöhen MTBF und zu verringern MTTR sowie dieMTTD.

Die Beseitigung aller Fehler wäre zwar ideal, aber nicht realistisch. In großen verteilten Systemen mit tief verteilten Abhängigkeiten werden Fehler auftreten. „Alles schlägt ständig fehl“ (siehe Werner Vogels, Amazon.comCTO, 10 Lektionen aus 10 Jahren Amazon Web Services.) und „Sie können keine Gesetze gegen Fehler erlassen [also] konzentrieren Sie sich auf schnelle Erkennung und Reaktion.“ (siehe Chris Pinkham, Gründungsmitglied des EC2 Amazon-Teams, ARC335Designing for failure: Architecting resilient systems on) AWS

Das bedeutet, dass Sie häufig keine Kontrolle darüber haben, ob es zu einem Ausfall kommt. Sie können kontrollieren, wie schnell Sie den Fehler erkennen und etwas dagegen unternehmen. Die Erhöhung der Verfügbarkeit MTBF ist zwar nach wie vor ein wichtiger Bestandteil der Hochverfügbarkeit, doch die wichtigsten Änderungen, auf die Kunden Einfluss haben, sind die Reduzierung MTTD undMTTR.

Themen

Reduzieren MTTD

Um die Anzahl MTTD der Fehler zu reduzieren, muss der Fehler so schnell wie möglich entdeckt werden. Die Verkürzung hängt MTTD von der Beobachtbarkeit ab, d. h. davon, wie Sie Ihren Workload instrumentiert haben, um seinen Status zu verstehen. Kunden sollten ihre Kennzahlen zur Kundenzufriedenheit in den kritischen Subsystemen ihrer Workloads überwachen, um proaktiv

zu erkennen, wann ein Problem auftritt (siehe Anhang 1). Weitere Informationen zu diesen Kennzahlen finden MTTD Sie auch bei MTTR kritischen Kennzahlen.). Kunden können Amazon CloudWatch Synthetics verwenden, um Kanarien zu erstellen, die Ihr System APIs und Ihre Konsolen überwachen, um die Benutzererfahrung proaktiv zu messen. Es gibt eine Reihe anderer Zustandsprüfungsmechanismen, die verwendet werden können, um diese zu minimierenMTTD, z. B. Elastic Load Balancing (ELB) -Zustandsprüfungen, Amazon Route 53-Zustandsprüfungen und mehr. (Siehe Amazon Builders' Library — Implementierung von Zustandsprüfungen.)

Ihre Überwachung muss auch in der Lage sein, Teilausfälle sowohl des Systems als Ganzes als auch Ihrer einzelnen Subsysteme zu erkennen. Ihre Verfügbarkeits-, Ausfall- und Latenzkennzahlen sollten die Dimensionalität Ihrer Fehlerisolationsgrenzen als CloudWatch metrische Dimensionen verwenden. Stellen Sie sich zum Beispiel eine einzelne EC2 Instanz vor, die Teil einer zellenbasierten Architektur ist, in der use1-az1 AZ, in der Region us-east-1, die Teil des Updates des Workloads ist, das Teil seines API Steuerebenen-Subsystems ist. Wenn der Server seine Messobjekte überträgt, kann er seine Instanz-ID, AZ, Region, API Namen und Subsystemnamen als Dimensionen verwenden. Auf diese Weise können Sie die Daten beobachten und Alarme für jede dieser Dimensionen einrichten, um Fehler zu erkennen.

Reduzieren MTTR

Nachdem ein Fehler entdeckt wurde, dient der Rest der MTTR Zeit der eigentlichen Reparatur oder Minderung der Auswirkungen. Um einen Fehler zu reparieren oder zu mindern, müssen Sie wissen, was falsch ist. Es gibt zwei Hauptgruppen von Kennzahlen, die in dieser Phase Aufschluss geben: 1/ Kennzahlen zur Folgenabschätzung und 2/ Kennzahlen zur betrieblichen Health. Die erste Gruppe gibt Aufschluss über das Ausmaß der Auswirkungen bei einem Ausfall. Dabei wird die Anzahl oder der Prozentsatz der betroffenen Kunden, Ressourcen oder Workloads gemessen. Die zweite Gruppe hilft bei der Identifizierung der Gründe für die Auswirkungen. Sobald das Warum erkannt wurde, können Bediener und die Automatisierung auf den Fehler reagieren und ihn beheben. Weitere Informationen zu diesen Kennzahlen finden Sie in Anhang 1 — MTTD und in MTTR wichtigen Kennzahlen.

> Regel 9
>
> Beobachtbarkeit und Instrumentierung sind entscheidend für die Reduzierung von MTTD undMTTR.

Umgehung von Ausfällen

Der schnellste Ansatz zur Minderung der Auswirkungen sind ausfallschnelle Subsysteme, die Ausfälle umgehen. Bei diesem Ansatz wird Redundanz verwendet, um das Problem zu reduzieren, MTTR indem die Arbeit eines ausgefallenen Subsystems schnell auf ein Ersatzsystem verlagert wird. Die Redundanz kann von Softwareprozessen über EC2 Instanzen bis hin zu mehreren Regionen AZs reichen.

Durch Ersatzsubsysteme kann die Geschwindigkeit auf nahezu MTTR Null reduziert werden. Die Wiederherstellungszeit ist nur das, was benötigt wird, um die Arbeit auf das Ersatzgerät umzuleiten. Dies geschieht häufig mit minimaler Latenz und ermöglicht es, die Arbeit innerhalb des definierten Zeitraums abzuschließenSLA, wobei die Verfügbarkeit des Systems erhalten bleibt. Dies führt MTTRs eher zu geringfügigen, vielleicht sogar unmerklichen Verzögerungen als zu längeren Zeiten der Nichtverfügbarkeit.

Wenn Ihr Service beispielsweise EC2 Instances hinter einem Application Load Balancer (ALB) verwendet, können Sie Integritätsprüfungen in einem Intervall von nur fünf Sekunden konfigurieren und nur zwei fehlgeschlagene Zustandsprüfungen benötigen, bevor ein Ziel als fehlerhaft markiert wird. Das bedeutet, dass Sie innerhalb von 10 Sekunden einen Fehler erkennen und das Senden von Datenverkehr an den fehlerhaften Host beenden können. In diesem Fall MTTR ist das praktisch dasselbe wie beim, MTTD da der Fehler, sobald er erkannt wird, auch behoben wird.

Genau das versuchen Workloads mit hoher Verfügbarkeit oder kontinuierlicher Verfügbarkeit zu erreichen. Wir möchten Ausfälle in der Arbeitslast schnell umgehen, indem wir ausgefallene Subsysteme schnell erkennen, sie als ausgefallen markieren, das Senden von Datenverkehr an sie beenden und stattdessen Datenverkehr an ein redundantes Subsystem weiterleiten.

Beachten Sie, dass Ihr Workload durch die Verwendung eines solchen Fail-Fast-Mechanismus sehr empfindlich auf vorübergehende Fehler reagiert. Stellen Sie im angegebenen Beispiel sicher, dass Ihre Load Balancer-Integritätsprüfungen nur oberflächliche Zustandsprüfungen oder Verfügbarkeitsprüfungen und lokale Integritätsprüfungen nur für die Instanz durchführen und keine Abhängigkeiten oder Workflows testen (oft als tiefgreifende Integritätsprüfungen bezeichnet). Dadurch wird verhindert, dass Instanzen bei vorübergehenden Fehlern, die sich auf die Arbeitslast auswirken, unnötig ausgetauscht werden.

Beobachtbarkeit und die Fähigkeit, Fehler in Subsystemen zu erkennen, sind entscheidend für die erfolgreiche Umgehung von Ausfällen. Sie müssen den Umfang der Auswirkungen kennen, damit die betroffenen Ressourcen als fehlerhaft oder ausgefallen markiert und außer Betrieb genommen

werden können, sodass sie umgeleitet werden können. Wenn beispielsweise eine einzelne AZ teilweise beeinträchtigt wird, muss Ihre Instrumentierung in der Lage sein, zu erkennen, dass ein AZ-lokales Problem vorliegt, damit alle Ressourcen in dieser AZ umgeleitet werden können, bis sie wiederhergestellt sind.

Für die Umgehung von Ausfällen durch Routing sind je nach Umgebung möglicherweise auch zusätzliche Tools erforderlich. Stellen Sie sich anhand des vorherigen Beispiels mit EC2 Instances hinter einem vorALB, dass Instanzen in einer AZ möglicherweise die lokalen Zustandsprüfungen bestehen, aber eine isolierte Beeinträchtigung der AZ dazu führt, dass sie keine Verbindung zu ihrer Datenbank in einer anderen AZ herstellen können. In diesem Fall werden diese Instanzen durch die Integritätsprüfungen des Lastenausgleichs nicht außer Betrieb genommen. Ein anderer automatisierter Mechanismus wäre erforderlich, um die AZ aus dem Load Balancer zu entfernen oder die Instances dazu zu zwingen, ihre Integritätsprüfungen nicht zu bestehen. Dies hängt davon ab, ob es sich bei dem Umfang der Auswirkungen um eine AZ handelt. Für Workloads, die keinen Load Balancer verwenden, wäre eine ähnliche Methode erforderlich, um zu verhindern, dass Ressourcen in einer bestimmten AZ Arbeitseinheiten akzeptieren oder Kapazität aus der AZ ganz entfernen.

In einigen Fällen kann die Verlagerung von Arbeit auf ein redundantes Subsystem nicht automatisiert werden, z. B. der Failover von einer primären auf eine sekundäre Datenbank, bei der die Technologie keine eigene Wahl zum Marktführer vorsieht. Dies ist ein übliches Szenario für Architekturen mit AWS mehreren Regionen. Da diese Arten von Failovers eine gewisse Ausfallzeit erfordern, nicht sofort rückgängig gemacht werden können und die Arbeitslast für einen gewissen Zeitraum ohne Redundanz bleibt, ist es wichtig, dass ein Mensch in den Entscheidungsprozess einbezogen wird.

Workloads, für die ein weniger striktes Konsistenzmodell verwendet werden kann, können kürzere Ergebnisse erzielen, MTTRs indem Failover-Automatisierung für mehrere Regionen verwendet wird, um Ausfälle zu umgehen. Funktionen wie die regionsübergreifende Amazon S3 S3-Replikation oder globale Amazon DynamoDB-Tabellen bieten Funktionen für mehrere Regionen durch letztlich konsistente Replikation. Darüber hinaus ist die Verwendung eines lockeren Konsistenzmodells von Vorteil, wenn wir das Theorem berücksichtigen. CAP Wenn der Workload bei Netzwerkausfällen, die sich auf die Konnektivität zu statusbehafteten Subsystemen auswirken, Verfügbarkeit der Konsistenz vorzieht, kann er dennoch fehlerfreie Antworten liefern — eine weitere Möglichkeit, Fehler zu umgehen.

Das Routing zur Umgehung von Ausfällen kann mit zwei verschiedenen Strategien implementiert werden. Die erste Strategie besteht in der Implementierung statischer Stabilität, indem im Voraus genügend Ressourcen bereitgestellt werden, um die gesamte Last des ausgefallenen Subsystems zu bewältigen. Dabei kann es sich um eine einzelne EC2 Instanz oder um eine gesamte Kapazität

von AZ handeln. Der Versuch, während eines Fehlers neue Ressourcen bereitzustellen, erhöht die Abhängigkeit von einer Kontrollebene in Ihrem Wiederherstellungspfad MTTR und fügt eine Abhängigkeit von dieser hinzu. Dies ist jedoch mit zusätzlichen Kosten verbunden.

Die zweite Strategie besteht darin, einen Teil des Datenverkehrs vom ausgefallenen Teilsystem auf andere umzuleiten und den überschüssigen Verkehr, der von der verbleibenden Kapazität nicht bewältigt werden kann, abzuleiten. Während dieser Phase der Verschlechterung können Sie neue Ressourcen aufstocken, um die ausgefallene Kapazität zu ersetzen. Dieser Ansatz dauert länger MTTR und führt zu einer Abhängigkeit von einer Steuerungsebene, kostet aber weniger, wenn Reservekapazitäten im Standby-Modus vorhanden sind.

Kehren Sie zu einem zweifelsfrei funktionierenden Zustand zurück

Ein weiterer gängiger Ansatz zur Risikominderung während einer Reparatur besteht darin, die Arbeitslast wieder in einen als funktionierend bekannten Zustand zu versetzen. Wenn eine kürzlich durchgeführte Änderung den Fehler verursacht haben könnte, ist das Zurücksetzen dieser Änderung eine Möglichkeit, zum vorherigen Zustand zurückzukehren.

In anderen Fällen könnten vorübergehende Bedingungen den Fehler verursacht haben. In diesem Fall könnte ein Neustart der Arbeitslast die Auswirkungen abschwächen. Lassen Sie uns diese beiden Szenarien untersuchen.

Während einer Bereitstellung MTTR hängt die Minimierung von MTTD und von Beobachtbarkeit und Automatisierung ab. Ihr Bereitstellungsprozess muss die Arbeitslast kontinuierlich auf erhöhte Fehlerraten, erhöhte Latenz oder Anomalien hin beobachten. Sobald diese erkannt wurden, sollte der Bereitstellungsprozess gestoppt werden.

Es gibt verschiedene Bereitstellungsstrategien, z. B. direkte Bereitstellungen, blaue/grüne Bereitstellungen und fortlaufende Bereitstellungen. Jede dieser Methoden verwendet möglicherweise einen anderen Mechanismus, um zu einem zweifelsfrei funktionierenden Zustand zurückzukehren. Es kann automatisch zum vorherigen Status zurückkehren, den Verkehr wieder in die blaue Umgebung verlagern oder ein manuelles Eingreifen erfordern.

CloudFormation bietet die Möglichkeit, im Rahmen seiner Stack-Erstellungs- und Aktualisierungsvorgänge ein automatisches Rollback durchzuführen, was AWS CodeDeployauch der Fall ist. CodeDeploy unterstützt auch Blau/Grün- und fortlaufende Bereitstellungen.

Um diese Funktionen zu nutzen und Ihren Energieverbrauch zu minimieren, sollten Sie erwägenMTTR, Ihre gesamte Infrastruktur- und Codebereitstellung mithilfe dieser Services zu

automatisieren. In Szenarien, in denen Sie diese Dienste nicht nutzen können, sollten Sie erwägen, das Saga-Muster zu implementieren, um fehlgeschlagene AWS Step Functions Bereitstellungen rückgängig zu machen.

Wenn Sie einen Neustart in Betracht ziehen, gibt es verschiedene Ansätze. Diese reichen vom Neustart eines Servers, der längsten Aufgabe, bis zum Neustart eines Threads, der kürzesten Aufgabe. In der folgenden Tabelle sind einige der Methoden zum Neustarten und die ungefähren Zeiten bis zum Abschluss aufgeführt (repräsentativ für Größenunterschiede, diese sind nicht exakt).

Mechanismus zur Behebung von Fehlern	Geschätzt MTTR
Starten und konfigurieren Sie einen neuen virtuellen Server	15 Minuten
Stellen Sie die Software erneut bereit	10 Minuten
Starten Sie den Server neu	5 Minuten
Starten Sie den Container neu oder starten Sie ihn	2 Sekunden
Rufen Sie eine neue serverlose Funktion auf	100 ms
Prozess neu starten	10 ms
Thread neu starten	10 µs

Wenn man sich die Tabelle ansieht, gibt es einige klare Vorteile MTTR bei der Verwendung von Containern und serverlosen Funktionen (wie AWS Lambda). Sie MTTR sind um Größenordnungen schneller als der Neustart einer virtuellen Maschine oder der Start einer neuen. Die Verwendung von Fehlerisolierung durch Softwaremodularität ist jedoch ebenfalls von Vorteil. Wenn Sie den Ausfall eines einzelnen Prozesses oder Threads eindämmen können, ist die Wiederherstellung nach diesem Fehler viel schneller als der Neustart eines Containers oder Servers.

Als allgemeine Methode zur Wiederherstellung können Sie von unten nach oben vorgehen: 1/ Restart, 2/Reboot, 3/Reimage/ReDeploy, 4/Replace. Sobald Sie jedoch mit dem Neustartschritt begonnen haben, ist das Routing zur Umgehung von Fehlern in der Regel schneller (in der Regel dauert es höchstens 3-4 Minuten). Um die Auswirkungen nach einem Neustartversuch so schnell

wie möglich abzumildern, umgehen Sie den Ausfall und setzen Sie dann im Hintergrund den Wiederherstellungsprozess fort, um die Kapazität Ihres Workloads wiederherzustellen.

> Regel 10
>
> Konzentrieren Sie sich auf die Minderung der Auswirkungen, nicht auf die Problemlösung. Gehen Sie den schnellsten Weg zurück zum Normalbetrieb.

Diagnose eines Fehlers

Ein Teil des Reparaturprozesses nach der Erkennung ist der Diagnosezeitraum. In diesem Zeitraum versuchen die Bediener herauszufinden, was falsch ist. Dieser Prozess kann das Abfragen von Protokollen, das Überprüfen von Operational Health-Metriken oder das Anmelden bei Hosts zur Fehlerbehebung beinhalten. All diese Aktionen benötigen Zeit. Daher kann die Erstellung von Tools und Runbooks zur Beschleunigung dieser Aktionen auch dazu beitragen, diesen Aufwand zu reduzieren. MTTR

Runbooks und Automatisierung

In ähnlicher Weise müssen Bediener, nachdem Sie festgestellt haben, was falsch ist und welche Maßnahmen zur Behebung der Arbeitslast ergriffen werden können, in der Regel einige Schritte ausführen, um das Problem zu lösen. Nach einem Ausfall kann die Arbeitslast beispielsweise am schnellsten repariert werden, indem sie neu gestartet wird, was mehrere, geordnete Schritte umfassen kann. Die Verwendung eines Runbooks, das diese Schritte entweder automatisiert oder einem Bediener spezifische Anweisungen gibt, beschleunigt den Vorgang und trägt dazu bei, das Risiko unbeabsichtigter Aktionen zu verringern.

Zunehmend MTBF

Die letzte Komponente zur Verbesserung der Verfügbarkeit ist die Erhöhung derMTBF. Dies kann sowohl für die Software als auch für die AWS Dienste gelten, mit denen sie ausgeführt wird.

Zunehmendes verteiltes System MTBF

Eine Möglichkeit zur Steigerung MTBF besteht darin, Fehler in der Software zu reduzieren. Dazu stehen verschiedene Möglichkeiten zur Verfügung. Kunden können Tools wie Amazon CodeGuru

Reviewer verwenden, um häufig auftretende Fehler zu finden und zu beheben. Sie sollten außerdem umfassende Peer-Code-Reviews, Komponententests, Integrationstests, Regressionstests und Auslastungstests an Software durchführen, bevor sie in der Produktion eingesetzt wird. Wenn Sie den Umfang der Codeabdeckung in Tests erhöhen, können Sie sicherstellen, dass auch ungewöhnliche Codeausführungspfade getestet werden.

Die Implementierung kleinerer Änderungen kann auch dazu beitragen, unerwartete Ergebnisse zu vermeiden, indem die Komplexität von Änderungen reduziert wird. Jede Aktivität bietet die Möglichkeit, Fehler zu identifizieren und zu beheben, bevor sie überhaupt in Anspruch genommen werden können.

Ein weiterer Ansatz zur Vermeidung von Ausfällen sind regelmäßige Tests. Durch die Implementierung eines Chaos Engineering-Programms können Sie testen, wie Ihr Workload ausfällt, Wiederherstellungsverfahren validieren und Fehlerquellen finden und beheben, bevor sie in der Produktion auftreten. Kunden können den AWS Fault Injection Simulator als Teil ihres Toolsets für Chaos-Engineering-Experimente verwenden.

Fehlertoleranz ist eine weitere Möglichkeit, Ausfälle in einem verteilten System zu verhindern. Fail-Fast-Module, Wiederholungsversuche mit exponentiellem Backoff und Jitter, Transaktionen und Idempotenz sind alles Techniken, um Workloads fehlertolerant zu machen.

Bei Transaktionen handelt es sich um eine Gruppe von Vorgängen, die den Eigenschaften entsprechen. ACID Diese sind:

- Atomarität — Entweder werden alle Aktionen ausgeführt oder keine von ihnen wird ausgeführt.
- Konsistenz — Bei jeder Transaktion wird der Workload in einem gültigen Zustand belassen.
- Isolation — Bei gleichzeitig ausgeführten Transaktionen bleibt der Workload in demselben Zustand, als ob sie sequentiell ausgeführt worden wären.
- Dauerhaftigkeit — Sobald eine Transaktion festgeschrieben ist, bleiben alle ihre Auswirkungen erhalten, auch wenn der Workload ausfällt.

Wiederholungen mit exponentiellem Backoff und Jitter ermöglichen es Ihnen, vorübergehende Ausfälle zu überwinden, die durch Heisenbugs, Überlastung oder andere Bedingungen verursacht werden. Wenn Transaktionen idempotent sind, können sie ohne Nebenwirkungen mehrfach wiederholt werden.

Wenn wir die Auswirkungen eines Heisenbugs auf eine fehlertolerante Hardwarekonfiguration berücksichtigen, wären wir ziemlich unbesorgt, da die Wahrscheinlichkeit, dass der Heisenbug

sowohl auf dem primären als auch auf dem redundanten Subsystem auftritt, verschwindend gering ist. (Siehe Jim Gray, „Warum stoppen Computer und was kann dagegen getan werden? „, Juni 1985, Technischer Tandembericht 85.7.) In verteilten Systemen wollen wir mit unserer Software dieselben Ergebnisse erzielen.

Wenn ein Heisenbug ausgelöst wird, ist es unerlässlich, dass die Software die fehlerhafte Operation schnell erkennt und fehlschlägt, damit sie erneut versucht werden kann. Dies wird durch defensive Programmierung und Validierung von Eingaben, Zwischenergebnissen und Ausgaben erreicht. Darüber hinaus sind Prozesse isoliert und teilen sich keinen Status mit anderen Prozessen.

Dieser modulare Ansatz stellt sicher, dass der Umfang der Auswirkungen bei einem Ausfall begrenzt ist. Prozesse schlagen unabhängig voneinander fehl. Wenn ein Prozess fehlschlägt, sollte die Software „Prozesspaare“ verwenden, um die Arbeit erneut zu versuchen, was bedeutet, dass ein neuer Prozess die Arbeit eines fehlgeschlagenen Prozesses übernehmen kann. Um die Zuverlässigkeit und Integrität der Arbeitslast zu gewährleisten, sollte jeder Vorgang als Transaktion behandelt werden. ACID

Auf diese Weise kann ein Prozess fehlschlagen, ohne dass der Status der Arbeitslast dadurch beeinträchtigt wird, dass die Transaktion abgebrochen und alle vorgenommenen Änderungen rückgängig gemacht werden. Auf diese Weise kann der Wiederherstellungsprozess die Transaktion aus einem zweifelsfrei funktionierenden Zustand erneut versuchen und anschließend ordnungsgemäß neu starten. Auf diese Weise kann Software fehlertolerant gegenüber Heisenbugs sein.

Sie sollten jedoch nicht darauf abzielen, Software fehlertolerant gegenüber Bohrbugs zu machen. Diese Fehler müssen gefunden und behoben werden, bevor die Arbeitslast in die Produktion übergeht, da keine Redundanz jemals zu einem korrekten Ergebnis führen kann. (Siehe Jim Gray, „Warum stoppen Computer und was kann dagegen getan werden? „, Juni 1985, Technischer Tandembericht 85.7.)

Der letzte Weg zur Erhöhung MTBF besteht darin, den Umfang der Auswirkungen eines Fehlers zu verringern. Die Verwendung von Fehlerisolierung durch Modularisierung zur Erstellung von Fehlercontainern ist hierfür eine der wichtigsten Methoden, wie weiter oben unter Fehlertoleranz und Fehlerisolierung beschrieben. Die Reduzierung der Ausfallrate verbessert die Verfügbarkeit. AWS verwendet Techniken wie die Aufteilung von Diensten in Steuerungsebenen und Datenebenen, Availability Zone Independence (AZI), regionale Isolierung, zellenbasierte Architekturen und Shuffle-Sharding zur Fehlerisolierung. Dies sind auch Muster, die auch von Kunden verwendet werden können. AWS

Schauen wir uns zum Beispiel ein Szenario an, in dem Kunden aufgrund eines Workloads in verschiedene Fehlercontainer der Infrastruktur aufgeteilt wurden, von denen höchstens 5% aller Kunden bedient wurden. Bei einem dieser Fehlercontainer tritt bei 10% der Anfragen ein Ereignis auf, das die Latenz über das Client-Timeout hinaus erhöht. Während dieses Ereignisses war der Service für 95% der Kunden zu 100% verfügbar. Bei den anderen 5% schien der Service zu 90% verfügbar zu sein. Dies führt zu einer Verfügbarkeit von 1 − (5% o f c u s t o m e r s × 10% o f t h e i r r r e q u e s t s) = 99,5% anstatt dass 10% der Anfragen bei 100% der Kunden fehlschlagen (was zu einer Verfügbarkeit von 90% führt).

> Regel 11
>
> Die Fehlerisolierung verringert den Umfang der Auswirkungen und erhöht die MTBF Arbeitslast, da die Gesamtausfallrate reduziert wird.

Zunehmende Abhängigkeit MTBF

Die erste Methode, um Ihre AWS Abhängigkeit zu erhöhen, MTBF ist die Verwendung von Fehlerisolierung. Viele AWS Dienste bieten ein gewisses Maß an Isolation in der AZ, was bedeutet, dass ein Ausfall in einer AZ keine Auswirkungen auf den Service in einer anderen AZ hat.

Die Verwendung redundanter EC2 Instanzen in mehreren Instanzen AZs erhöht die Verfügbarkeit des Subsystems. AZIbietet eine sparsame Kapazität innerhalb einer einzelnen Region, sodass Sie Ihre Verfügbarkeit für AZI Dienste erhöhen können.

Allerdings funktionieren nicht alle AWS Dienste auf AZ-Ebene. Viele andere bieten regionale Isolation. In diesem Fall, in dem die Verfügbarkeit des Regionaldienstes nicht die für Ihre Arbeitslast erforderliche Gesamtverfügbarkeit unterstützt, könnten Sie einen regionsübergreifenden Ansatz in Betracht ziehen. Jede Region bietet eine isolierte Instanziierung des Dienstes, was einem Sparing entspricht.

Es gibt verschiedene Dienste, die den Aufbau eines Dienstes für mehrere Regionen erleichtern. Beispielsweise:

- Globale Amazon Aurora Aurora-Datenbank
- Globale Amazon DynamoDB-Tabellen
- Amazon ElastiCache (RedisOSS) — Globaler Datenspeicher
- AWS Globaler Beschleuniger

- Regionsübergreifende Amazon S3 S3-Replikation
- Amazon Route 53-Controller für die Anwendungswiederherstellung

Dieses Dokument befasst sich nicht mit den Strategien für den Aufbau von Workloads in mehreren Regionen. Sie sollten jedoch die Verfügbarkeitsvorteile von Architekturen mit mehreren Regionen gegen die zusätzlichen Kosten, die Komplexität und die betrieblichen Abläufe abwägen, die erforderlich sind, um Ihre gewünschten Verfügbarkeitsziele zu erreichen.

Die nächste Methode zur Erhöhung der Abhängigkeit MTBF besteht darin, Ihren Workload so zu gestalten, dass er statisch stabil ist. Beispiel: Sie haben einen Workload, der Produktinformationen bereitstellt. Wenn Ihre Kunden eine Anfrage für ein Produkt stellen, sendet Ihr Service eine Anfrage an einen externen Metadatendienst, um Produktdetails abzurufen. Dann gibt Ihr Workload all diese Informationen an den Benutzer zurück.

Wenn der Metadatendienst jedoch nicht verfügbar ist, schlagen die Anfragen Ihrer Kunden fehl. Stattdessen können Sie die Metadaten asynchron lokal in Ihren Service abrufen oder per Push übertragen, um Anfragen zu beantworten. Dadurch entfällt der synchrone Aufruf des Metadatendienstes aus Ihrem kritischen Pfad.

Da Ihr Service auch dann noch verfügbar ist, wenn der Metadaten-Service nicht verfügbar ist, können Sie ihn bei Ihrer Verfügbarkeitsberechnung als Abhängigkeit entfernen. Dieses Beispiel basiert auf der Annahme, dass sich die Metadaten nicht häufig ändern und dass die Bereitstellung veralteter Metadaten besser ist als die fehlgeschlagene Anfrage. Ein anderes ähnliches Beispiel ist serve-stale, da es ermöglichtDNS, Daten über das TTL Ablaufdatum hinaus im Cache zu behalten und für Antworten zu verwenden, wenn eine aktualisierte Antwort nicht sofort verfügbar ist.

Die letzte Methode zur Erhöhung der Abhängigkeit MTBF besteht darin, den Umfang der Auswirkungen eines Fehlers zu verringern. Wie bereits erwähnt, handelt es sich bei einem Ausfall nicht um ein binäres Ereignis, sondern um verschiedene Ausfälle. Dies ist der Effekt der Modularisierung; Fehler beschränken sich nur auf die Anfragen oder Benutzer, die von diesem Container bedient werden.

Dies führt zu weniger Ausfällen während eines Ereignisses, was letztlich die Verfügbarkeit der gesamten Arbeitslast erhöht, da der Umfang der Auswirkungen begrenzt wird.

Reduzierung häufiger Einflussquellen

1985 entdeckte Jim Gray im Rahmen einer Studie bei Tandem Computers, dass Misserfolge hauptsächlich auf zwei Faktoren zurückzuführen waren: Software und Betrieb. (Siehe Jim Gray,

„Warum stoppen Computer und was kann dagegen getan werden? „, Juni 1985, Technischer Tandembericht 85.7.) Dies gilt auch nach 36 Jahren noch immer. Trotz technologischer Fortschritte gibt es keine einfache Lösung für diese Probleme, und die Hauptausfallquellen haben sich nicht geändert. Die Behebung von Softwarefehlern wurde zu Beginn dieses Abschnitts erörtert, weshalb der Schwerpunkt hier auf dem Betrieb und der Reduzierung der Fehlerhäufigkeit liegen wird.

Stabilität im Vergleich zu Funktionen

Wenn wir uns in diesem Abschnitt die Grafik mit den Ausfallraten für Software und Hardware ansehenthe section called “Verfügbarkeit verteilter Systeme”, können wir feststellen, dass in jeder Softwareversion Fehler hinzukommen. Das bedeutet, dass jede Änderung der Arbeitslast ein erhöhtes Ausfallrisiko mit sich bringt. Bei diesen Änderungen handelt es sich in der Regel um Dinge wie neue Funktionen, was eine logische Folge darstellt. Workloads mit höherer Verfügbarkeit werden die Stabilität gegenüber neuen Funktionen begünstigen. Daher besteht eine der einfachsten Möglichkeiten zur Verbesserung der Verfügbarkeit darin, seltener bereitzustellen oder weniger Funktionen bereitzustellen. Workloads, die häufiger bereitgestellt werden, weisen naturgemäß eine geringere Verfügbarkeit auf als Workloads, bei denen dies nicht der Fall ist. Workloads, bei denen keine Funktionen hinzugefügt werden können, halten jedoch nicht mit der Kundennachfrage Schritt und können mit der Zeit an Nutzen verlieren.

Wie können wir also weiterhin innovativ sein und Funktionen sicher veröffentlichen? Die Antwort lautet Standardisierung. Was ist die richtige Art der Bereitstellung? Wie ordnet man Bereitstellungen an? Was sind die Standards für Tests? Wie lange wartest du zwischen den Stufen? Decken Ihre Komponententests den Softwarecode ausreichend ab? Diese Fragen werden durch Standardisierung beantwortet und Probleme vermieden, die beispielsweise dadurch entstehen, dass keine Lasttests durchgeführt werden, Bereitstellungsphasen übersprungen werden oder zu schnell auf zu vielen Hosts bereitgestellt werden.

Die Art und Weise, wie Sie Standardisierung implementieren, erfolgt durch Automatisierung. Es reduziert die Wahrscheinlichkeit menschlicher Fehler und ermöglicht Computern, das zu tun, worin sie gut sind, nämlich jedes Mal dasselbe immer und immer wieder auf die gleiche Weise zu tun. Die Art und Weise, wie man Standardisierung und Automatisierung zusammenhält, besteht darin, sich Ziele zu setzen. Ziele wie keine manuellen Änderungen, Hostzugriff nur über bedingte Autorisierungssysteme, das Schreiben von Lasttests für alle API und so weiter. Operative Exzellenz ist eine kulturelle Norm, die tiefgreifende Veränderungen erfordern kann. Die Festlegung und Verfolgung der Leistung anhand eines Ziels trägt dazu bei, einen kulturellen Wandel voranzutreiben, der weitreichende Auswirkungen auf die Verfügbarkeit von Workloads haben wird. Die Säule AWS Well-Architected Operational Excellence bietet umfassende Best Practices für operative Exzellenz.

Sicherheit für den Bediener

Der andere Hauptverursacher von betrieblichen Ereignissen, die zu Ausfällen führen, sind Menschen. Menschen machen Fehler. Sie verwenden möglicherweise die falschen Anmeldeinformationen, geben den falschen Befehl ein, drücken zu früh die Eingabetaste oder verpassen einen wichtigen Schritt. Manuelles Handeln führt immer wieder zu Fehlern, was wiederum zum Scheitern führt.

Eine der Hauptursachen für Bedienfehler sind verwirrende, wenig intuitive oder inkonsistente Benutzeroberflächen. Jim Gray stellte in seiner Studie von 1985 auch fest, dass „Benutzeroberflächen, die den Bediener nach Informationen fragen oder ihn auffordern, eine Funktion auszuführen, einfach, konsistent und fehlertolerant sein müssen“. (Siehe Jim Gray, „Warum hören Computer auf und was kann dagegen getan werden? „, Juni 1985, Technischer Tandembericht 85.7.) Diese Einsicht gilt auch heute noch. In den letzten drei Jahrzehnten gab es in der Branche zahlreiche Beispiele, bei denen eine verwirrende oder komplexe Benutzeroberfläche, fehlende Bestätigungen oder Anweisungen oder auch einfach nur eine unfreundliche menschliche Sprache dazu geführt haben, dass ein Bediener das Falsche getan hat.

> **Regel 12**
>
> Machen Sie es den Bedienern leicht, das Richtige zu tun.

Vermeidung von Überlastung

Der letzte gemeinsame Einflussfaktor sind Ihre Kunden, die eigentlichen Nutzer Ihres Workloads. Erfolgreiche Workloads werden in der Regel häufig genutzt, aber manchmal übersteigt diese Nutzung die Skalierbarkeit des Workloads. Es gibt viele Dinge, die passieren können: Festplatten können voll werden, Thread-Pools könnten erschöpft sein, die Netzwerkbandbreite könnte ausgelastet sein oder es können Grenzwerte für Datenbankverbindungen erreicht werden.

Es gibt keine ausfallsichere Methode, um diese zu beseitigen, aber die proaktive Überwachung von Kapazität und Auslastung anhand von Operational Health Metriken gibt frühzeitig Warnmeldungen, wenn diese Ausfälle auftreten könnten. Techniken wie Load-Shedding, Schutzschalter und Wiederholungsversuche mit exponentiellem Backoff und Jitter können dazu beitragen, die Auswirkungen zu minimieren und die Erfolgsquote zu erhöhen, aber diese Situationen stellen immer noch einen Fehler dar. Automatisierte Skalierung auf der Grundlage von Operational Health-Metriken kann dazu beitragen, die Häufigkeit von Ausfällen aufgrund von Überlastung zu reduzieren, ist aber möglicherweise nicht in der Lage, schnell genug auf Nutzungsänderungen zu reagieren.

Wenn Sie die kontinuierlich verfügbare Kapazität für Kunden sicherstellen möchten, müssen Sie Kompromisse bei Verfügbarkeit und Kosten eingehen. Eine Möglichkeit, um sicherzustellen, dass Kapazitätsmangel nicht zu Nichtverfügbarkeit führt, besteht darin, jedem Kunden ein Kontingent zuzuweisen und sicherzustellen, dass die Kapazität Ihres Workloads so skaliert wird, dass 100% der zugewiesenen Kontingente bereitgestellt werden. Wenn Kunden ihr Kontingent überschreiten, werden sie gedrosselt. Das ist kein Fehler und wird nicht auf die Verfügbarkeit angerechnet. Sie müssen auch Ihren Kundenstamm genau verfolgen und die future Auslastung prognostizieren, um ausreichend Kapazität bereitzustellen. Auf diese Weise wird sichergestellt, dass Ihr Workload nicht aufgrund einer zu hohen Auslastung durch Ihre Kunden zu Ausfallszenarien gezwungen wird.

- Amazon Builders' Library — Verwendung von Load Shedding zur Vermeidung von Überlastung
- Amazon Builders' Library — Fairness in Mehrmandantensystemen

Schauen wir uns zum Beispiel einen Workload an, der einen Speicherservice bereitstellt. Jeder Server im Workload kann 100 Downloads pro Sekunde unterstützen, Kunden erhalten ein Kontingent von 200 Downloads pro Sekunde und es gibt 500 Kunden. Um dieses Kundenvolumen unterstützen zu können, muss der Service Kapazität für 100.000 Downloads pro Sekunde bereitstellen, wofür 1.000 Server erforderlich sind. Wenn ein Kunde sein Kontingent überschreitet, wird er gedrosselt, sodass für jeden anderen Kunden ausreichend Kapazität zur Verfügung steht. Dies ist ein einfaches Beispiel für eine Möglichkeit, Überlastung zu vermeiden, ohne Arbeitseinheiten abzulehnen.

Schlussfolgerung

In diesem Dokument haben wir 12 Regeln für hohe Verfügbarkeit festgelegt.

- Regel 1 — Weniger häufige Ausfälle (längere MTBF), kürzere Ausfallerkennungszeiten (kürzere MTTD) und kürzere Reparaturzeiten (kürzere MTTR) sind die drei Faktoren, die zur Verbesserung der Verfügbarkeit in verteilten Systemen genutzt werden.
- Regel 2 — Die Verfügbarkeit der Software in Ihrem Workload ist ein wichtiger Faktor für die Gesamtverfügbarkeit Ihres Workloads und sollte genauso berücksichtigt werden wie andere Komponenten.
- Regel 3 — Die Reduzierung von Abhängigkeiten kann sich positiv auf die Verfügbarkeit auswirken.
- Regel 4 — Wählen Sie im Allgemeinen Abhängigkeiten aus, deren Verfügbarkeitsziele den Zielen Ihres Workloads entsprechen oder diese übertreffen.
- Regel 5 — Verwenden Sie Sparing, um die Verfügbarkeit von Abhängigkeiten in einem Workload zu erhöhen.
- Regel 6 — Es gibt eine Obergrenze für die Kosteneffizienz des Sparens. Verwenden Sie die wenigsten Ersatzteile, die erforderlich sind, um die erforderliche Verfügbarkeit zu erreichen.
- Regel 7 — Machen Sie keine Abhängigkeiten von den Steuerungsebenen in Ihrer Datenebene, insbesondere bei der Wiederherstellung.
- Regel 8 — Verknüpfen Sie Abhängigkeiten lose, damit Ihr Workload nach Möglichkeit trotz Beeinträchtigung der Abhängigkeiten ordnungsgemäß ausgeführt werden kann.
- Regel 9 — Beobachtbarkeit und Instrumentierung sind entscheidend für die Reduzierung von MTTD und MTTR.
- Regel 10 — Konzentrieren Sie sich auf die Minderung der Auswirkungen, nicht auf die Problemlösung. Nehmen Sie den schnellsten Weg zurück zum Normalbetrieb.
- Regel 11 — Die Fehlerisolierung verringert den Umfang der Auswirkungen und erhöht die MTBF des Workloads, indem die Gesamtausfallrate reduziert wird.
- Regel 12 — Machen Sie es den Bedienern leicht, das Richtige zu tun.

Die Verbesserung der Workload-Verfügbarkeit wird durch die Reduzierung von MTTD und MTTR sowie die Erhöhung des MTBF erreicht. Zusammenfassend haben wir die folgenden Möglichkeiten zur Verbesserung der Verfügbarkeit erörtert, die sich auf Technologie, Mitarbeiter und Prozesse beziehen.

- MTTD
  - Reduzieren Sie die MTTD durch eine proaktive Überwachung Ihrer Kundenerlebniskennzahlen.
  - Nutzen Sie detaillierte Zustandsprüfungen für ein schnelles Failover.
- MTTR
  - Überwachen Sie den Umfang der Auswirkungen und die Kennzahlen zur betrieblichen Gesundheit.
  - Reduzieren Sie die MTTR, indem Sie 1/Restart, 2/Reboot, 3/Re-Image/Redeploy und 4/Replace befolgen.
  - Umgehen Sie Fehler, indem Sie den Umfang der Auswirkungen verstehen.
  - Nutzen Sie Dienste mit schnelleren Neustartzeiten wie Container und serverlose Funktionen über virtuelle Maschinen oder physische Hosts.
  - Automatisches Rollback fehlgeschlagener Bereitstellungen, sofern möglich.
  - Richten Sie Runbooks und Betriebstools für Diagnosevorgänge und Neustartverfahren ein.
- MTBF
  - Beseitigen Sie Bugs und Defekte in der Software durch strenge Tests, bevor sie für die Produktion freigegeben werden.
  - Implementieren Sie Chaos-Engineering und Fault Injection.
  - Verwenden Sie die richtige Menge an Abhängigkeiten, um Fehler zu tolerieren.
  - Minimieren Sie den Umfang der Auswirkungen bei Ausfällen mithilfe von Fehlercontainern.
  - Implementieren Sie Standards für Bereitstellungen und Änderungen.
  - Entwerfen Sie einfache, intuitive, konsistente und gut dokumentierte Bedienoberflächen.
  - Setzen Sie sich Ziele für betriebliche Exzellenz.
  - Bevorzugen Sie Stabilität gegenüber der Veröffentlichung neuer Funktionen, wenn Verfügbarkeit eine kritische Dimension Ihres Workloads ist.
  - Implementieren Sie Nutzungskontingente mit Drosselung oder Lastabwurf oder beidem, um eine Überlastung zu vermeiden.

Denken Sie daran, dass es uns niemals vollständig gelingen wird, Misserfolge zu verhindern. Konzentrieren Sie sich auf Softwaredesigns mit der bestmöglichen Ausfallisolierung, die Umfang und Ausmaß der Auswirkungen begrenzt, und halten Sie diese Auswirkungen idealerweise unter den Schwellenwerten für „Ausfallzeiten“ UND investieren Sie in eine sehr schnelle, sehr zuverlässige

Erkennung und Abwehr. Moderne verteilte Systeme müssen Ausfälle immer noch als unvermeidlich ansehen und auf allen Ebenen auf hohe Verfügbarkeit ausgelegt werden.

Anhang 1 — Kritische MTTD- und MTTR-Metriken

Im Folgenden finden Sie einen Rahmen für die Standardisierung von Instrumentierung und Beobachtbarkeit, der dazu beitragen kann, die MTTD und MTTR während eines Ereignisses zu reduzieren.

Kennzahlen zum Kundenerlebnis. Diese Kennzahlen zeigen, dass ein Service reaktionsschnell und verfügbar ist, um Kundenanfragen zu bearbeiten. Zum Beispiel die Latenz auf der Steuerungsebene. Diese Metriken messen die Fehlerrate, Verfügbarkeit, Latenz, Lautstärke und Drosselungsrate.

Metriken zur Folgenabschätzung. Diese Kennzahlen geben Aufschluss über den Umfang der Auswirkungen von Ereignissen. Zum Beispiel die Anzahl oder der Prozentsatz der Kunden, die von einem Ereignis auf der Datenebene betroffen sind. Misst die Anzahl oder den Prozentsatz der betroffenen Dinge.

Kennzahlen zur betrieblichen Gesundheit. Diese Kennzahlen spiegeln wider, dass ein Service auf Kundenanfragen reagiert und verfügbar ist, sich jedoch auf gemeinsame Infrastruktursubsysteme und -ressourcen konzentriert. Zum Beispiel der prozentuale Anteil der CPU-Auslastung Ihrer EC2-Flotte. Diese Metriken sollten Auslastung, Kapazität, Durchsatz, Fehlerrate, Verfügbarkeit und Latenz messen.

Beitragende Faktoren

Zu den Mitwirkenden an diesem Dokument gehören:

- Michael Haken, Principal Solutions Architect, Amazon Web Services

Weitere Informationen

Weitere Informationen finden Sie unter:

- Eine gut durchdachte Zuverlässigkeitssäule
- Gut konzipierte Säule für betriebliche Exzellenz
- Amazon Builders' Library — Gewährleistung der Rollback-Sicherheit bei Bereitstellungen
- Amazon Builders' Library — Beyond five 9s: Lehren aus unseren höchsten verfügbaren Datenebenen
- Amazon Builders' Library — Automatisierung sicherer, praktischer Bereitstellungen
- Amazon Builders' Library — Architektur und Betrieb robuster serverloser Systeme im großen Maßstab
- Amazon Builders' Library — Amazons Ansatz für die Bereitstellung hoher Verfügbarkeit
- Amazon Builders' Library — Amazons Ansatz zum Aufbau stabiler Dienste
- Amazon Builders' Library — Amazons Ansatz, um erfolgreich zu scheitern
- AWSZentrum für Architektur

Dokumentverlauf

Abonnieren Sie den RSS-Feed, um über Aktualisierungen dieses Whitepapers informiert zu werden.

Änderung	Beschreibung	Datum
Erstveröffentlichung	Das Whitepaper wurde zuerst veröffentlicht.	12. November 2021

> **Note**
>
> Um RSS-Updates zu abonnieren, müssen Sie ein RSS-Plug-In für den von Ihnen verwendeten Browser aktiviert haben.

Hinweise

Die Kunden sind dafür verantwortlich, die Informationen in diesem Dokument selbst unabhängig zu bewerten. Dieses Dokument: (a) dient nur zu Informationszwecken, (b) stellt aktuelle AWS Produktangebote und Praktiken dar, die ohne vorherige Ankündigung geändert werden können, und (c) enthält keine Verpflichtungen oder Zusicherungen von AWS und seinen verbundenen Unternehmen, Lieferanten oder Lizenzgebern. AWSProdukte oder Dienstleistungen werden „wie sie sind“ ohne ausdrückliche oder stillschweigende Garantien, Zusicherungen oder Bedingungen jeglicher Art bereitgestellt. Die Verantwortlichkeiten und Verbindlichkeiten AWS gegenüber seinen Kunden werden durch AWS Vereinbarungen geregelt, und dieses Dokument ist weder Teil einer Vereinbarung zwischen AWS seinen Kunden noch ändert es diese.

AWS-Glossar

Die neueste AWS-Terminologie finden Sie im AWS-Glossar in der AWS-Glossar-Referenz.

Die vorliegende Übersetzung wurde maschinell erstellt. Im Falle eines Konflikts oder eines Widerspruchs zwischen dieser übersetzten Fassung und der englischen Fassung (einschließlich infolge von Verzögerungen bei der Übersetzung) ist die englische Fassung maßgeblich.