AWS Documento técnico

Disponibilidad y más allá: comprender y mejorar la resiliencia de los sistemas distribuidos en AWS

Disponibilidad y más allá: comprender y mejorar la resiliencia de los sistemas distribuidos en AWS: AWS Documento técnico

Table of Contents

Resumen e introducción i
Introducción 1
Descripción de la disponibilidad 2
Disponibilidad de los sistemas distribuidos 4
Tipos de errores en los sistemas distribuidos 7
Disponibilidad con dependencias 7
Disponibilidad con redundancia 9
Teorema CAP 14
Tolerancia a errores y aislamiento de errores 15
Cómo medir la disponibilidad 18
Índice de éxito de las solicitudes del servidor y del cliente 18
Tiempo de inactividad anual 21
Latencia 22
Diseñar sistemas distribuidos de alta disponibilidad en AWS 23
Reduciendo MTTD 23
Reduciendo MTTR 24
Cómo sortear los errores 25
Vuelta a un estado válido conocido 27
Diagnóstico de errores 29
Manuales de procedimientos y automatización 29
Aumentando MTBF 29
Incrementar el sistema distribuido MTBF 29
Dependencia creciente MTBF 32
Cómo reducir las fuentes de impacto más habituales 33
Conclusión 37
Apéndice 1: Métricas fundamentales del MTTD y el MTTR 40
Colaboradores 41
Documentación adicional 42
Historial de documentos 43
Avisos 44
Glosario de AWS 45
..... xlvi

Disponibilidad y más allá: Descripción y mejora de la resiliencia de los sistemas distribuidos en AWS

Fecha de publicación: 12 de noviembre de 2021 (Historial de documentos)

En la actualidad, las empresas utilizan sistemas complejos y distribuidos tanto en la nube como en las instalaciones. Quieren que estas cargas de trabajo sean resilientes para poder atender a sus clientes y cumplir sus objetivos empresariales. Este documento describe comúnmente la disponibilidad como medida de la resiliencia, define reglas para crear cargas de trabajo de alta disponibilidad y ofrece orientación sobre cómo mejorar la disponibilidad de las cargas de trabajo.

Introducción

¿Qué significa crear una carga de trabajo de alta disponibilidad? ¿Cómo se mide la disponibilidad? ¿Qué puedo hacer para aumentar la disponibilidad de mi carga de trabajo? Este documento le ayudará a responder a este tipo de preguntas. Está dividido en tres secciones principales. La primera sección, Descripción de la disponibilidad, es en gran parte teórica. Establece una definición común para la disponibilidad y los factores que la afectan. La segunda sección, Cómo medir la disponibilidad, proporciona orientación sobre cómo medir empíricamente la disponibilidad de una carga de trabajo. La tercera sección, Cómo diseñar sistemas distribuidos de alta disponibilidad en AWS, es una aplicación práctica de las ideas presentadas en la primera sección. Además, a lo largo de estas secciones, este documento identificará las reglas para crear cargas de trabajo resilientes. El objetivo de este documento es respaldar la orientación y las prácticas recomendadas presentadas en el Pilar de fiabilidad del marco AWS Well-Architected Framework.

A lo largo de todo el contenido del documento, encontrarás muchas expresiones matemáticas algebraicas. Las conclusiones clave son los conceptos que respaldan estas matemáticas, no las matemáticas en sí mismas. Dicho esto, presentar un desafío también es la intención de este documento. Cuando se utilizan cargas de trabajo de alta disponibilidad, es necesario poder demostrar matemáticamente que lo que se ha creado está consiguiendo su objetivo. Es posible que incluso los mejores diseños basados en buenas intenciones no logren el resultado deseado de manera coherente. Esto significa que se necesitan mecanismos que midan la eficacia de la solución y, por lo tanto, se necesita cierto nivel de matemáticas para crear y operar sistemas distribuidos resilientes y de alta disponibilidad.

Descripción de la disponibilidad

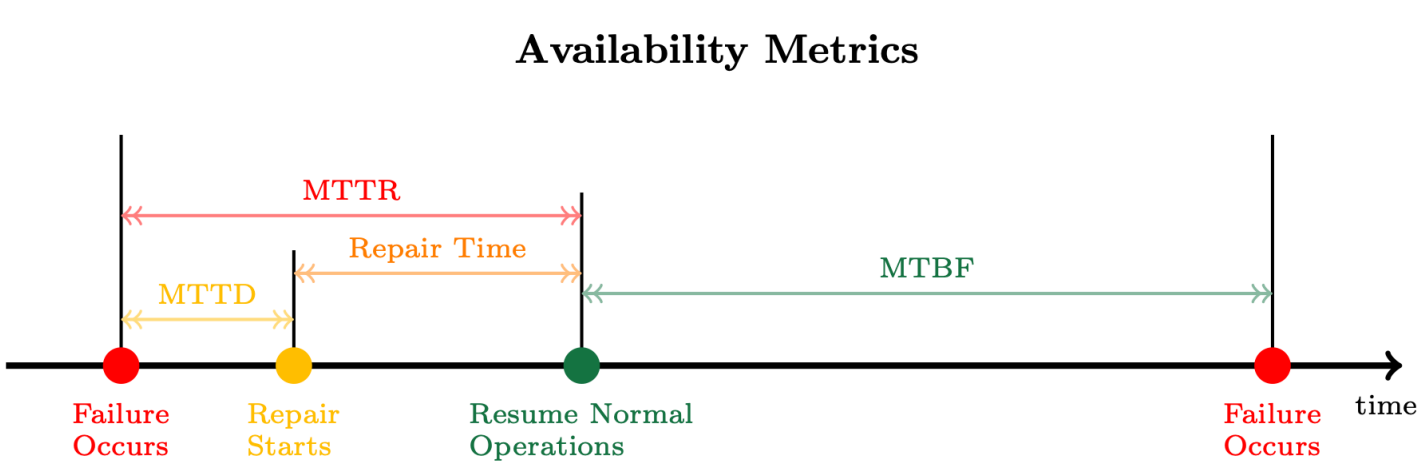
La disponibilidad es una de las principales formas en que podemos medir cuantitativamente la resiliencia. Definimos la disponibilidad (D) como el porcentaje de tiempo que una carga de trabajo está disponible para utilizarse. Es una relación entre el tiempo de actividad esperado (estar disponible) y el tiempo total que se está midiendo (el tiempo de actividad esperado más el tiempo de inactividad esperado).

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

Ecuación 1: Disponibilidad

Para entender mejor esta fórmula, veremos cómo medir el tiempo de actividad y el tiempo de inactividad. En primer lugar, queremos saber cuánto durará la carga de trabajo sin que se produzca ningún error. A esto lo denominamos tiempo medio entre errores (MTBF), que es el tiempo medio que transcurre entre el momento en que una carga de trabajo comienza a funcionar con normalidad y el siguiente error. Luego, queremos saber cuánto tardará en recuperarse la carga de trabajo tras un error.

A esto lo denominamos tiempo medio de reparación (o recuperación) (MTTR), que es el período de tiempo en el que la carga de trabajo no está disponible mientras se repara el subsistema averiado o este empieza a funcionar de nuevo. Un período de tiempo importante en el MTTR es el tiempo medio de detección (MTTD); es decir, el tiempo que transcurre entre el momento en que ocurre el error y el inicio de las operaciones de reparación. En el siguiente diagrama se muestra cómo se relacionan todas estas métricas.



Relación entre el MTTD, el MTTR y el MTBF

De este modo, podemos expresar la disponibilidad, (D) con el MTBF (el tiempo que la carga de trabajo está activa) y el MTTR (el tiempo en que la carga de trabajo está inactiva).

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

Ecuación 2: Relación entre el MTBF y el MTTR

Y la probabilidad de que la carga de trabajo esté inactiva (es decir, no disponible) es la probabilidad de que falle (F).

$$F = 1 - A$$

Ecuación 3: Probabilidad de error

La fiabilidad es la capacidad de una carga de trabajo para hacer lo correcto, cuando se le solicita, dentro del tiempo de respuesta especificado. Esto es lo que mide la disponibilidad. El hecho de que una carga de trabajo falle con menos frecuencia (MTBF más largo) o que tenga un tiempo de reparación o recuperación más corto (MTTR más corto) mejora su disponibilidad.

Regla 1

Los tres factores que se utilizan para mejorar la disponibilidad en los sistemas distribuidos son una menor frecuencia de errores (MTBF más largo), tiempos de detección de errores más cortos (MTTD más corto) y tiempos de reparación más cortos (MTTR más corto).

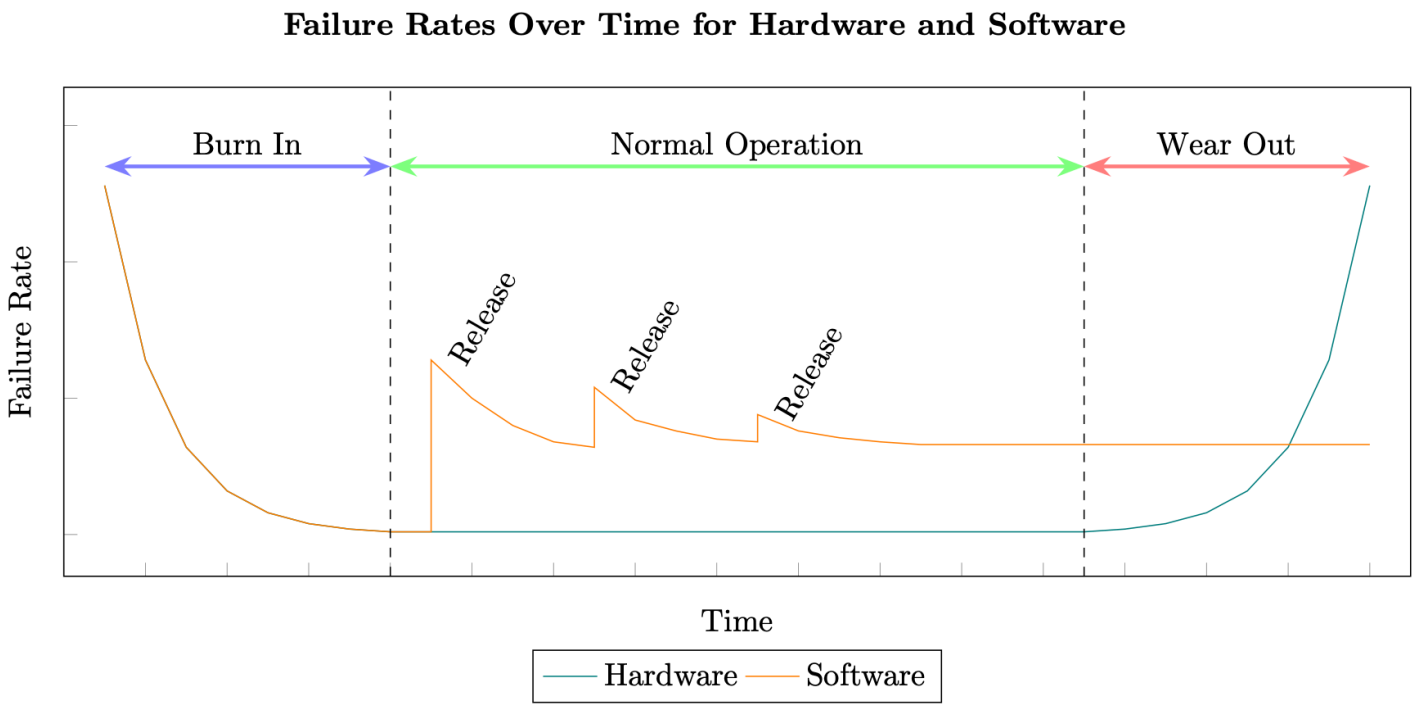
Temas

- Disponibilidad de los sistemas distribuidos
- Disponibilidad con dependencias
- Disponibilidad con redundancia
- Teorema CAP
- Tolerancia a errores y aislamiento de errores

Disponibilidad de los sistemas distribuidos

Los sistemas distribuidos se componen tanto de componentes de software como de componentes de hardware. Algunos de los componentes de software podrían ser en sí mismos otro sistema distribuido. La disponibilidad de los componentes de hardware y software subyacentes afecta a la disponibilidad resultante de la carga de trabajo.

El cálculo de la disponibilidad mediante el tiempo medio entre errores (MTBF) y el tiempo medio de reparación o recuperación (MTTR) tiene sus raíces en los sistemas de hardware. Sin embargo, los sistemas distribuidos fallan por motivos muy diferentes a los de un componente de hardware. Mientras que un fabricante puede calcular de manera coherente el tiempo promedio antes de que se deteriore un componente de hardware, las mismas pruebas no se pueden aplicar a los componentes de software de un sistema distribuido. El software suele seguir la curva en forma de bañera (donde la curva desciende primero bruscamente, luego se produce un periodo de estabilidad y, por último, la curva asciende de manera drástica), mientras que el software sigue una curva escalonada producida por los defectos adicionales que se van introduciendo con cada nueva versión (consulte Fiabilidad del software).



Índices de hardware y software

Además, el software de los sistemas distribuidos suele cambiar a índices exponencialmente superiores a las del hardware. Por ejemplo, un disco duro magnético estándar puede tener un índice medio de errores anual (AFR) del 0,93 %, lo que, en la práctica, en el caso de un disco HDD puede suponer una vida útil de al menos 3 a 5 años antes de que llegue al período de deterioro, y podría ser superior (consulte los datos y estadísticas sobre discos duros publicados en 2020 por Backblaze). El disco duro no cambia sustancialmente durante esa vida útil, mientras que, en un plazo de 3 a 5 años, por ejemplo, Amazon podría implementar de 450 a 750 millones de cambios en sus sistemas de software (consulte Amazon Builders' Library: Automatización de implementaciones seguras y sin intervención).

El hardware también está sujeto al concepto de obsolescencia programada; es decir, tiene una vida útil incorporada y será necesario reemplazarlo después de un período de tiempo determinado. (Lea "The Great Lightbulb Conspiracy"). En teoría, el software no está sujeto a esta restricción, no tiene período de deterioro alguno y puede funcionar indefinidamente.

Todo esto significa que los mismos modelos de prueba y predicción utilizados para calcular el MTBF y el MTTR del hardware no se aplican al software. Desde la década de 1970, se han realizado cientos de intentos de diseñar modelos para solucionar este problema, pero generalmente todos

se dividen en dos categorías: modelos de predicción y modelos de estimación (consulte la lista de modelos de fiabilidad del software).

Según esto, el cálculo de un MTBF y un MTTR prospectivos para sistemas distribuidos y, por lo tanto, de una disponibilidad prospectiva, siempre se derivará de algún tipo de predicción o estimación. Pueden generarse mediante modelos predictivos, simulaciones estocásticas, análisis históricos o pruebas rigurosas, pero esos cálculos no garantizan el tiempo de actividad o el tiempo de inactividad.

Es posible que las razones por las que un sistema distribuido falló en el pasado nunca vuelvan a repetirse. Es probable que las causas por las que se produzca un error en el futuro sean diferentes y, posiblemente, desconocidas. Los mecanismos de recuperación necesarios también pueden ser diferentes para errores futuros de los utilizados en el pasado y requerir mucho más o menos tiempo.

Además, los MTBF y MTTR son promedios. Habrá alguna variación entre el valor promedio y los valores reales observados (la desviación estándar [σ] mide esta variación). Por lo tanto, es posible que trascurra más o menos tiempo entre errores de las cargas de trabajo y que los tiempos de recuperación sean algo diferentes en un entorno de producción real.

Dicho esto, la disponibilidad de los componentes de software que componen un sistema distribuido sigue siendo importante. El software puede fallar por numerosos motivos (que se analizarán con más detalle en la siguiente sección) y repercuten en la disponibilidad de la carga de trabajo. Por lo tanto, en el caso de los sistemas distribuidos de alta disponibilidad, se debe prestar la misma atención al cálculo, la medición y la mejora de la disponibilidad de los componentes de software que a los subsistemas de hardware y software externos.

> **Regla 2**
>
> La disponibilidad del software en su carga de trabajo es un factor importante de la disponibilidad general de la carga de trabajo y debe recibir la misma atención que otros componentes.

Es importante señalar que, a pesar de que el MTBF y el MTTR son difíciles de predecir para los sistemas distribuidos, aún proporcionan información clave sobre cómo mejorar la disponibilidad. Reducir la frecuencia de los errores (mayor MTBF) y disminuir el tiempo de recuperación después de que se produzca un error (menor MTTR) redundará en una mayor disponibilidad empírica.

Tipos de errores en los sistemas distribuidos

En general, existen dos clases de errores en los sistemas distribuidos que afectan a la disponibilidad, denominados cariñosamente Bohrbug y Heisenbug (lea "A Conversation with Bruce Lindsay" de ACM Queue; vol. 2, núm. 8; noviembre de 2004).

Un Bohrbug es un problema de software funcional que se repite. Con los mismos datos de entrada, el error generará coherentemente los mismos datos de salida incorrectos (como el modelo atómico determinista de Bohr, que es sólido y se detecta fácilmente). Estos tipos de errores son poco frecuentes cuando una carga de trabajo entra en el entorno de producción.

Un Heisenbug es un error transitorio, lo que significa que solo ocurre en condiciones específicas e infrecuentes. Estas condiciones suelen estar relacionadas con aspectos como el hardware (por ejemplo, un error transitorio del dispositivo o detalles específicos de la implementación del hardware, como el tamaño del registro), optimizaciones del compilador e implementación del lenguaje, condiciones límite (por ejemplo, falta de almacenamiento temporal) o condiciones de carrera (por ejemplo, no usar un semáforo para operaciones con varios subprocesos).

Los errores Heisenbug constituyen la mayoría de los errores de producción y son difíciles de encontrar porque son esquivos y parecen cambiar de comportamiento o desaparecer cuando se intenta observarlos o depurarlos. Sin embargo, si se reinicia el programa, es probable que la operación fallida se realice correctamente porque el entorno operativo es ligeramente diferente, lo que elimina las condiciones que originaron el Heisenbug.

Por lo tanto, la mayoría de los errores en el entorno de producción son transitorios y, cuando se vuelve a intentar realizar la operación, es poco probable que vuelva a fallar. Para ser resilientes, los sistemas distribuidos tienen que ser tolerantes a los errores Heisenbug. Descubriremos cómo se logra esto en la sección Cómo incrementar el MTBF de los sistemas distribuidos.

Disponibilidad con dependencias

En la sección anterior, mencionamos que el hardware, el software y, potencialmente, otros sistemas distribuidos son todos componentes de la carga de trabajo. A estos componentes los denominamos dependencias, los elementos de los que depende una carga de trabajo para proporcionar su funcionalidad. Existen las dependencias fuertes, que son aquellos elementos sin los que la carga de trabajo no puede funcionar, y dependencias suaves, cuya falta de disponibilidad puede pasar desapercibida o tolerarse durante un período de tiempo. Las dependencias fuertes repercuten directamente en la disponibilidad de la carga de trabajo.

Podríamos intentar calcular la disponibilidad máxima teórica de una carga de trabajo. Se trata del producto de la disponibilidad de todas las dependencias, incluido el propio software (α_n es la disponibilidad de un único subsistema), ya que todas ellas deben estar operativas.

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

Ecuación 4: Disponibilidad máxima teórica

Los números de disponibilidad utilizados en estos cálculos suelen estar asociados a aspectos como los acuerdos de nivel de servicio (SLA) o los objetivos de nivel de servicio (SLO). Los SLA definen el nivel de servicio esperado que recibirán los clientes, las métricas con las que se mide el servicio y las soluciones o penalizaciones (normalmente monetarias) en caso de que no se cumplan los niveles de servicio.

Con la fórmula anterior, podemos concluir que, desde un punto de vista puramente matemático, una carga de trabajo no puede estar más disponible que alguna de sus dependencias. Pero no coincide con lo que solemos ver. Una carga de trabajo creada con dos o tres dependencias con un SLA de disponibilidad del 99,99 % puede lograr una disponibilidad del 99,99% o superior por sí misma.

Esto se debe a que, como explicamos en la sección anterior, estas cifras de disponibilidad son estimaciones. Estiman o predicen la frecuencia con la que se produce un error y la rapidez con la que se puede reparar. No garantizan el tiempo de inactividad. Las dependencias suelen superar el SLA o el SLO de disponibilidad indicados.

Las dependencias también pueden tener objetivos de disponibilidad interna más altos con respecto a los objetivos de rendimiento que las cifras proporcionadas en los SLA públicos. Esto ofrece un cierto nivel de mitigación de riesgos a la hora de cumplir los SLA cuando ocurre algo desconocido o impredecible.

Por último, es posible que su carga de trabajo tenga dependencias cuyos SLA no puedan conocerse o que no ofrezcan un SLA o un SLO. Por ejemplo, el enrutamiento de Internet en todo el mundo es una dependencia común para muchas cargas de trabajo, pero es difícil saber qué proveedores de servicios de Internet utiliza su tráfico global, si tienen algún SLA o si los proveedores funcionan de forma coherente entre sí.

Lo que todo esto nos indica es que calcular una disponibilidad teórica máxima solo es probable que produzca un cálculo aproximado, pero por sí mismo es probable que no sea preciso ni proporcione información significativa Lo que sí nos dicen las matemáticas es que de cuantas menos cosas

dependa su carga de trabajo, menor probabilidad habrá de que se produzca, en general, un error. Cuantos menos números menores de uno se multipliquen, mayor será el resultado.

> **Regla 3**
>
> Reducir las dependencias puede tener un impacto positivo en la disponibilidad.

Las matemáticas también ayudan a conformar el proceso de selección de dependencias. El proceso de selección afecta a la forma de diseñar la carga de trabajo, la manera de aprovechar la redundancia en las dependencias para mejorar su disponibilidad y a decidir si esas dependencias son fuertes o suaves. Las dependencias que pueden afectar a la carga de trabajo se deben elegir cuidadosamente. La siguiente regla proporciona orientación sobre cómo hacerlo.

> **Regla 4**
>
> En general, seleccione las dependencias cuyos objetivos de disponibilidad sean iguales o superiores a los objetivos de su carga de trabajo.

Disponibilidad con redundancia

Cuando una carga de trabajo utiliza subsistemas múltiples, independientes y redundantes, puede alcanzar un mayor nivel teórico de disponibilidad que si se utilizara un único subsistema. Piense en una carga de trabajo compuesta de dos subsistemas idénticos. Puede estar completamente operativa si el subsistema uno o el subsistema dos están operativos. Para que todo el sistema esté inactivo, ambos subsistemas deben estar inactivos al mismo tiempo.

Si la probabilidad de error de un subsistema es $1 - \alpha$, entonces la probabilidad de que dos subsistemas redundantes estén inactivos al mismo tiempo es el producto de la probabilidad de que falle cada subsistema, $F = (1 - \alpha_1) \times (1 - \alpha_2)$. Para una carga de trabajo con dos subsistemas redundantes, si se utiliza la ecuación (3), se obtiene una disponibilidad definida de la siguiente manera:

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

Ecuación 5

Por lo tanto, para dos subsistemas cuya disponibilidad es del 99 %, la probabilidad de que uno falle es del 1 % y la probabilidad de que ambos fallen es (1 − 99 %) × (1 − 99 %) = 0,01 %. Así, la disponibilidad al usar dos subsistemas redundantes es del 99,99 %.

Esto se puede generalizar para incorporar otros componentes redundantes de repuesto, r. En la ecuación (5), solo asumimos un repuesto, pero una carga de trabajo puede tener dos, tres o más repuestos para poder sobrevivir a la pérdida simultánea de varios subsistemas sin afectar a la disponibilidad. Si una carga de trabajo tiene tres subsistemas y dos son de repuesto, la probabilidad de que los tres subsistemas fallen al mismo tiempo es (1 − α) × (1 − α) × (1 − α) o $(1 - \alpha)^3$. En general, una carga de trabajo con r repuestos solo fallará si fallan los subsistemas s + 1.

Para una carga de trabajo con n subsistemas y r repuestos, f es el número de modos de error o las formas en que los subsistemas r + 1 pueden fallar a partir de n.

En efecto, este es el teorema binomial, la matemática combinatoria de elegir k elementos de un conjunto de n, o “n elige k”. En este caso, k es r+ 1.

$$k = s + 1$$

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

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

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

Ecuación 6

Luego, podemos producir una aproximación generalizada de disponibilidad que incorpore el número de modos de error y los repuestos. (Para entender por qué hablamos de una aproximación, consulte el apéndice 2 de Highleyman, et al., "Breaking the Availability Barrier").

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

Ecuación 7

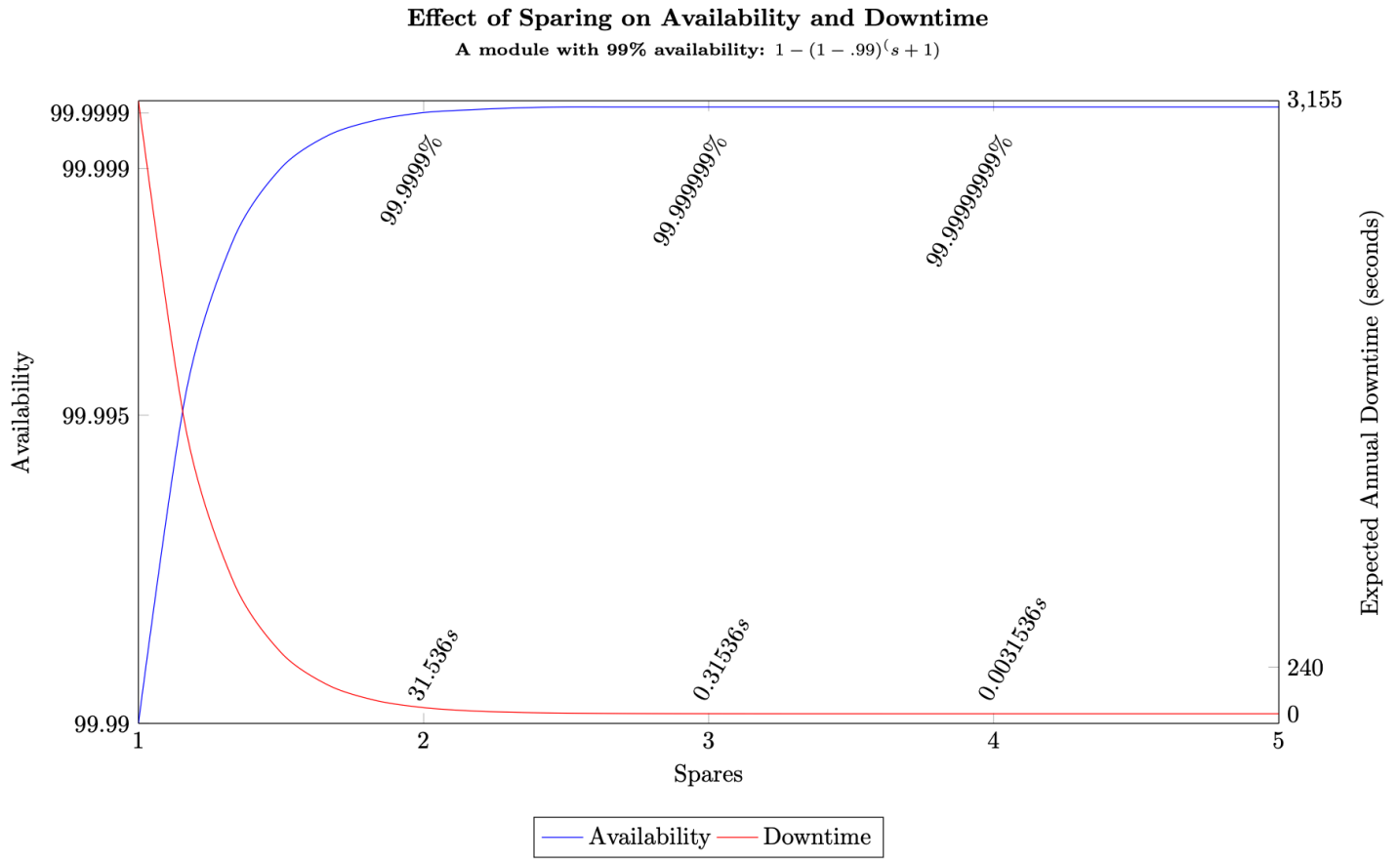
La incorporación de repuestos se puede aplicar a cualquier dependencia que proporcione recursos que fallen de forma independiente. Entre los ejemplos se incluyen instancias de Amazon EC2 en diferentes zonas de disponibilidad (AZ) o buckets de Amazon S3 en diferentes Regiones de AWS. El uso de repuestos ayuda a esa dependencia a lograr una mayor disponibilidad total para cumplir con los objetivos de disponibilidad de la carga de trabajo.

> **Regla 5**
>
> Utilice repuestos para incrementar la disponibilidad de las dependencias en una carga de trabajo.

Sin embargo, el uso de repuestos conlleva un costo. Cada repuesto adicional cuesta lo mismo que el módulo original, lo que incrementa el gasto de manera lineal. Crear una carga de trabajo que pueda utilizar componentes de repuesto también aumenta su complejidad. Debe saber cómo identificar los errores de dependencias, trasladar el trabajo a un recurso en buen estado y gestionar la capacidad general de la carga de trabajo.

La redundancia supone un problema de optimización. Si hay pocos repuestos, la carga de trabajo puede fallar con más frecuencia de la deseada; si hay demasiados, la carga de trabajo cuesta demasiado. Existe un umbral en el que añadir más repuestos costará más de lo que justifica la disponibilidad adicional que ofrecen.

Si utilizamos nuestra fórmula de disponibilidad general con repuestos, la ecuación (7), para un subsistema que tiene una disponibilidad del 99,5 %, con dos repuestos, la disponibilidad de la carga de trabajo es $D \approx 1 - (1)(1 - 0{,}995)^3 = 99{,}9999875\ \%$ (aproximadamente 3,94 segundos de inactividad al año); con 10 repuestos obtenemos $D \approx 1 - (1)(1 - 0{,}995)^{11} = 99{,}99999999999999999999$ (el tiempo de inactividad aproximado sería $1{,}26252 \times 10^{-15}$ ms al año, prácticamente 0). Al comparar estas dos cargas de trabajo, hemos incurrido en un aumento de 5 veces en el costo de los repuestos para lograr cuatro segundos menos de tiempo de inactividad al año. Para la mayoría de las cargas de trabajo, el aumento del costo no estaría justificado para este aumento de la disponibilidad. La relación se muestra en la siguiente figura.



Rendimientos decrecientes derivados del aumento de repuestos

Con tres repuestos o más, el resultado son fracciones de segundo del tiempo de inactividad previsto al año, lo que significa que, a partir de esa cantidad, se llega a la zona de rentabilidad decreciente. Puede que surja la necesidad de añadir más capacidad para lograr niveles más altos de disponibilidad, pero en realidad, la rentabilidad desaparece muy rápidamente. El uso de más de tres repuestos no proporciona ganancias materiales ni notables para casi ninguna carga de trabajo cuando el propio subsistema en sí tiene una disponibilidad de al menos el 99 %.

> **Regla 6**
>
> La rentabilidad del uso de repuestos tiene un límite superior. Utilice el menor número de repuestos necesario para lograr la disponibilidad requerida.

Debe tener en cuenta la unidad de error al seleccionar el número correcto de repuestos. Tomemos como ejemplo una carga de trabajo que requiere 10 instancias de EC2 para gestionar los picos de capacidad, implementadas en una sola AZ.

Dado que las AZ se han diseñado para funcionar como límites de aislamiento de errores, la unidad de error no es solo una instancia de EC2, sino que todas las instancias de EC2 de una AZ pueden fallar al mismo tiempo. En este caso, querrá añadir redundancia con otra AZ e implementar 10 instancias de EC2 adicionales para gestionar la carga en caso de que se produzca un error en la AZ, lo que supone un total de 20 instancias de EC2 (siguiendo el patrón de estabilidad estática).

Si bien parece que hablamos de 10 instancias de EC2 de reserva o de repuesto, en realidad se trata de una única AZ de reserva, por lo que no hemos superado el punto de rentabilidad decreciente. Sin embargo, puede disfrutar de más rentabilidad y, al mismo tiempo, incrementar la disponibilidad si utiliza tres AZ e implementa cinco instancias de EC2 por AZ.

Esto proporciona una AZ de reserva con un total de 15 instancias de EC2 (frente a dos AZ con 20 instancias) y, al mismo tiempo, proporciona las 10 instancias necesarias en total para atender los picos de capacidad durante un evento que afecte a una única AZ. Por lo tanto, debe incorporar repuestos para tolerar los errores en todos los límites de aislamiento de errores utilizados por la carga de trabajo (instancia, celda, AZ y región).

Teorema CAP

Otra manera de entender la disponibilidad guarda relación con el teorema CAP. Este teorema establece que un sistema distribuido, formado por varios nodos que almacenan datos, no puede ofrecer simultáneamente más de dos de las siguientes tres garantías:

- Coherencia (consistency): cada solicitud de lectura recibe la escritura más reciente o un error cuando no se puede garantizar la coherencia.
- Disponibilidad (availability): todas las solicitudes reciben una respuesta sin errores, aunque los nodos no estén disponibles ni activos.
- Tolerancia a las particiones (partition tolerance): el sistema sigue funcionando a pesar de la pérdida de un número arbitrario de mensajes entre los nodos.

(Para obtener más información, lea a Seth Gilbert y Nancy Lynch en “Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services”; ACM SIGACT News; volumen 33, número 2 (2002), págs. 51—59).

La mayoría de los sistemas distribuidos tienen que tolerar los errores de la red y, por lo tanto, se debe permitir la partición de la red. Esto significa que estas cargas de trabajo deben elegir entre la coherencia y la disponibilidad cuando se produce una partición de red. Si la carga de trabajo elige

la disponibilidad, siempre devuelve una respuesta, pero con datos potencialmente incoherentes. Si elige la coherencia, durante una partición de red devolverá un error, ya que la carga de trabajo no puede garantizar la coherencia de los datos.

Para las cargas de trabajo cuyo objetivo es proporcionar niveles de disponibilidad más elevados, pueden elegir la disponibilidad y la tolerancia a las particiones para evitar que se produzcan errores (no estar disponible) durante una partición de red. Esto hace que se requiera un modelo de coherencia más relajado, como coherencia final o coherencia monótona.

Tolerancia a errores y aislamiento de errores

Estos son dos conceptos importantes cuando pensamos en la disponibilidad. La tolerancia a errores es la capacidad de resistir los errores de los subsistemas y mantener la disponibilidad (hacer lo correcto dentro de un acuerdo de nivel de servicio [SLA] establecido). Para implementar la tolerancia a errores, las cargas de trabajo utilizan subsistemas de repuesto, de reserva o redundantes. Cuando uno de los subsistemas de un conjunto redundante falla, otro retoma su trabajo, por lo general casi sin problemas. En este caso, los repuestos son realmente capacidad de reserva; están disponibles para asumir el 100 % del trabajo del subsistema con errores. Cuando hablamos de verdaderos repuestos, es necesario que se produzcan varios errores en el subsistema para que repercutan negativamente en la carga de trabajo.

El aislamiento de errores minimiza el alcance del impacto cuando se produce un error. Esto se suele implementar con la modularización. Las cargas de trabajo se dividen en pequeños subsistemas que fallan de forma independiente y se pueden reparar de forma aislada. Un error que se produce en un módulo no se propaga más allá del módulo. Esta idea se extiende tanto verticalmente, a través de distintas funcionalidades de una carga de trabajo, como horizontalmente, a través de varios subsistemas que proporcionan la misma funcionalidad. Estos módulos actúan como contenedores de errores que limitan el alcance del impacto durante un evento.

Los patrones arquitectónicos de los planos de control, los planos de datos y la estabilidad estática respaldan directamente la implementación de la tolerancia a errores y el aislamiento de errores. El artículo de Amazon Builders' Library Estabilidad estática con zonas de disponibilidad define bien estos términos y cómo se aplican a la creación de cargas de trabajo resilientes y de alta disponibilidad. Este documento técnico utiliza estos patrones en la sección Cómo diseñar sistemas distribuidos de alta disponibilidad en AWS, y también resumimos sus definiciones aquí.

- Plano de control: la parte de la carga de trabajo involucrada en realizar cambios. Agregar recursos, elimina recursos, modifica recursos y propaga los cambios donde sean necesarios. Los planos de

control suelen ser más complejos y tienen más partes móviles que los planos de datos, por lo que estadísticamente tienen más probabilidades de fallar y tener una menor disponibilidad.

- Plano de datos: la parte de la carga de trabajo que proporciona la funcionalidad empresarial diaria. Los planos de datos suelen ser más simples y funcionan a volúmenes más altos que los planos de control, lo que se traduce en una mayor disponibilidad.
- Estabilidad estática: la capacidad de una carga de trabajo para continuar funcionando correctamente a pesar de las deficiencias de las dependencias. Un método de implementación consiste en eliminar las dependencias del plano de control de los planos de datos. Otro método consiste en acoplar de forma flexible las dependencias de la carga de trabajo. Es posible que la carga de trabajo no vea ninguna información actualizada (como cosas nuevas, eliminadas o modificadas) que se supone que su dependencia debía haber proporcionado. Sin embargo, todo lo que hacía antes de que la dependencia se viera afectada sigue funcionando.

Cuando pensamos en el deterioro de una carga de trabajo, hay dos enfoques de alto nivel que podemos plantearnos para la recuperación. El primer método consiste en responder a ese deterioro después de que se produzca, tal vez recurriendo a AWS Auto Scaling para añadir capacidad nueva. El segundo método consiste en prepararse para esas deficiencias antes de que se produzcan, tal vez aprovisionando en exceso la infraestructura de una carga de trabajo para que pueda seguir funcionando correctamente sin necesidad de añadir recursos adicionales.

Un sistema estable desde el punto de vista estático utiliza este último enfoque. Aprovisiona la capacidad de reserva previamente para que esté disponible en caso de error. Este método evita crear una dependencia en un plano de control en la ruta de recuperación de la carga de trabajo para aprovisionar nueva capacidad para recuperarse de un error. Además, aprovisionar nueva capacidad para varios recursos lleva tiempo. Mientras espera nueva capacidad, la carga de trabajo puede verse sobrecargada por la demanda existente y sufrir una mayor degradación, lo que puede provocar una pérdida total de la disponibilidad. Sin embargo, también se deben comparar las implicaciones financieras de utilizar capacidad previamente aprovisionada con los objetivos de disponibilidad.

La estabilidad estática brinda las dos reglas siguientes para las cargas de trabajo de alta disponibilidad.

> **Regla 7**
>
> No tenga dependencias en los planos de control en el plano de datos, especialmente durante una recuperación.

Regla 8

Cuando sea posible, acople las dependencias de manera flexible para que la carga de trabajo pueda funcionar correctamente a pesar del deterioro de la dependencia.

Cómo medir la disponibilidad

Como hemos visto anteriormente, crear un modelo de disponibilidad prospectivo para un sistema distribuido es difícil y puede que no proporcione los resultados deseados. Lo que puede ser más útil es desarrollar formas coherentes de medir la disponibilidad de la carga de trabajo.

La definición de disponibilidad como tiempo de actividad y tiempo de inactividad representa el error como una opción binaria: o bien la carga de trabajo está activa o no lo está.

Sin embargo, esto rara vez es el caso. El error tiene cierto grado de impacto y, a menudo, se produce en algún subconjunto de la carga de trabajo y afecta a un porcentaje de usuarios o solicitudes, a un porcentaje de ubicaciones o a un percentil de latencia. Todos estos son modos de error parcial.

Y si bien el tiempo medio de recuperación (MTTR) y el tiempo medio entre errores (MTBF) son útiles para comprender qué es lo que impulsa la disponibilidad resultante de un sistema y, por lo tanto, cómo mejorarlo, su utilidad no representa una medida empírica de la disponibilidad. Además, las cargas de trabajo se componen de muchos elementos. Por ejemplo, una carga de trabajo tipo un sistema de procesamiento de pagos se compone de muchas interfaces de programación de aplicaciones (API) y subsistemas. Entonces, cuando nos preguntamos por la disponibilidad de toda la carga de trabajo, estamos haciendo una pregunta muy compleja y con matices.

En esta sección, vamos a examinar tres formas de medir empíricamente la disponibilidad: el índice de éxito de las solicitudes del servidor, el índice de éxito de las solicitudes del cliente y el tiempo de inactividad anual.

Índice de éxito de las solicitudes del servidor y del cliente

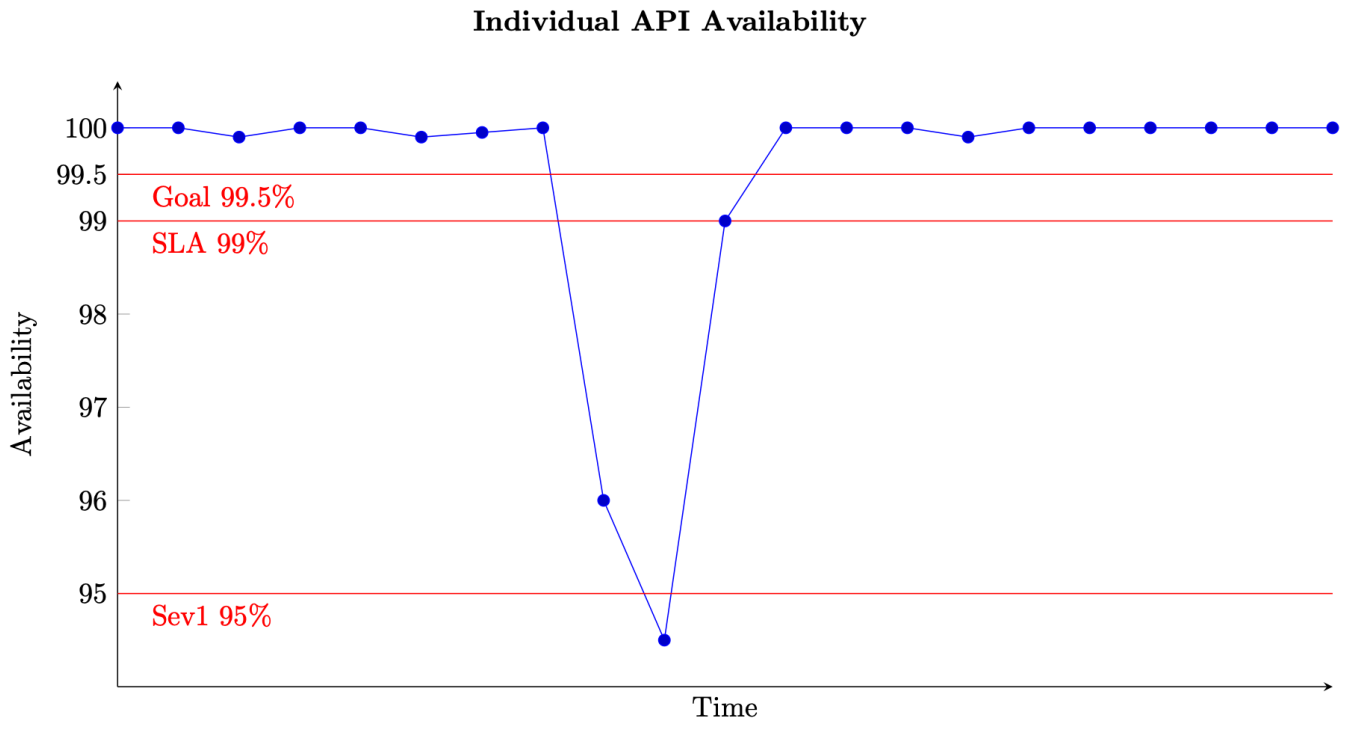
Los dos primeros métodos son muy similares y solo que difieren desde el punto de vista en que se realiza la medición. Las métricas del servidor se pueden recopilar a partir de la instrumentación del servicio. Sin embargo, no están completas. Si los clientes no pueden acceder al servicio, las métricas no podrán recopilarse. Para entender la experiencia del cliente, en lugar de confiar en la telemetría de los clientes para detectar solicitudes con errores, una forma más sencilla de recopilar métricas del cliente es simular el tráfico de clientes con valores controlados de un software que comprueba periódicamente los servicios y registra las métricas.

Estos dos métodos calculan la disponibilidad como la fracción del total de unidades de trabajo válidas que el servicio recibe y las que procesa correctamente (ignora las unidades de trabajo no válidas, como una solicitud HTTP que causa un error 404).

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

Ecuación 8

Para un servicio basado en solicitudes, la unidad de trabajo es la solicitud, como una solicitud HTTP. En el caso de los servicios basados en eventos o en tareas, las unidades de trabajo son eventos o tareas, como procesar un mensaje de una cola. Esta medida de disponibilidad es significativa en intervalos de tiempo cortos, como intervalos de uno o cinco minutos. También es más adecuada desde una perspectiva granular, por ejemplo, a nivel de API para un servicio basado en solicitudes. La siguiente figura proporciona una vista del aspecto que podría tener la disponibilidad a lo largo del tiempo si se calcula de esta manera. Cada punto de datos del gráfico se calcula mediante la ecuación (8) durante un período de cinco minutos (se pueden elegir otras dimensiones de tiempo, como intervalos de uno o diez minutos). Por ejemplo, el punto de datos 10 muestra una disponibilidad del 94,5 %. Esto significa que, entre los minutos t+45 y t+50, si el servicio recibió 1000 solicitudes, solo 945 de ellas se procesaron correctamente.



Ejemplo de medición de la disponibilidad a lo largo del tiempo para una sola API

El gráfico también muestra el objetivo de disponibilidad de la API (99,5 %), el acuerdo de nivel de servicio (SLA) que ofrece a los clientes en cuanto a la disponibilidad (99 %) y el umbral para que se envíe una alarma de alta gravedad (95 %). Sin el contexto de estos diferentes umbrales, es posible que un gráfico de disponibilidad no proporcione información significativa sobre el funcionamiento del servicio.

También queremos poder hacer un seguimiento de la disponibilidad de un subsistema más grande y describirla, como un plano de control, o de un servicio completo. Una forma de hacerlo es tomar el promedio de cada punto de datos de cinco minutos para cada subsistema. El gráfico tendrá un aspecto similar al anterior, pero representará un conjunto más grande de datos de entrada. También otorga el mismo peso a todos los subsistemas que componen el servicio. Un enfoque alternativo podría ser sumar todas las solicitudes recibidas y procesadas con éxito de todas las API en el servicio para calcular la disponibilidad en intervalos de cinco minutos.

Sin embargo, este último método podría ocultar una API individual con un bajo rendimiento y una baja disponibilidad. Tomemos como ejemplo sencillo un servicio con dos API.

La primera API recibe 1 000 000 de solicitudes en un período de cinco minutos y procesa correctamente 999 000 de ellas, lo que proporciona una disponibilidad del 99,9 %. La segunda API recibe 100 solicitudes en el mismo período de cinco minutos y procesa correctamente solamente 50 de ellas, lo que proporciona una disponibilidad del 50 %.

Si sumamos las solicitudes de ambas API, tenemos un total de 1 000 100 solicitudes válidas y 999 050 de ellas se procesaron correctamente, lo que supone una disponibilidad del 99,895 % para el servicio en general. Sin embargo, si calculamos la media de las disponibilidades de las dos API (el primer método), obtenemos una disponibilidad del 74,95 %, lo que podría reflejar mejor la experiencia real.

Ninguno de los dos enfoques es erróneo, pero demuestra la importancia de entender lo que indican las métricas de disponibilidad. Puede optar por sumar las solicitudes de todos los subsistemas si su carga de trabajo recibe un volumen de solicitudes similar en cada uno de ellos. Este enfoque se centra en la propia solicitud y en su éxito como medida de la disponibilidad y la experiencia del cliente. Como alternativa, puede optar por calcular la media de las disponibilidades de los subsistemas para representar de manera equitativa su criticidad, a pesar de las diferencias existentes en el volumen de solicitudes. Este enfoque se centra en los subsistemas y en la capacidad de cada uno de ellos como indicador de la experiencia del cliente.

Tiempo de inactividad anual

El tercer enfoque consiste en calcular el tiempo de inactividad anual. Esta forma de métrica de disponibilidad es más adecuada para establecer y revisar objetivos a largo plazo. Requiere definir qué significa el tiempo de inactividad para la carga de trabajo. Después, puede medir la disponibilidad en función del número de minutos que la carga de trabajo no estuvo en estado de interrupción en relación con el número total de minutos del período determinado.

Es posible que algunas cargas de trabajo puedan definir el tiempo de inactividad como una caída por debajo del 95 % de la disponibilidad de una sola API o función de la carga de trabajo durante un intervalo de uno o cinco minutos (lo que vimos en el gráfico de disponibilidad anterior). También puede considerar el tiempo de inactividad solo en lo que respecta a un subconjunto de operaciones críticas del plano de datos. Por ejemplo, el acuerdo de nivel de servicio de mensajería de Amazon (SQS, SNS) para la disponibilidad de SQS se aplica a la API de envío, recepción y eliminación de SQS.

Es posible que las cargas de trabajo más grandes y complejas deban definir las métricas de disponibilidad de todo el sistema. En el caso de un sitio de comercio electrónico de grandes dimensiones, una métrica para todo el sistema puede ser algo así como el volumen de pedidos de los clientes. En este caso, una caída del 10 % o más en los pedidos en comparación con la cantidad prevista durante un período de cinco minutos puede definir el tiempo de inactividad.

En cualquiera de los dos enfoques, se pueden sumar todos los períodos de interrupción para calcular la disponibilidad anual. Por ejemplo, si durante un año natural hubo 27 períodos de inactividad de cinco minutos, definidos como una disminución de la disponibilidad de cualquier API del plano de datos por debajo del 95 %, el tiempo de inactividad total fue de 135 minutos (algunos períodos de cinco minutos podrían haber sido consecutivos y otros aislados), lo que representa una disponibilidad anual del 99,97 %.

Este método adicional de medición de la disponibilidad puede proporcionar datos e información que no se encuentran ni en las métricas del cliente ni en las del servidor. Tomemos como ejemplo una carga de trabajo que no funciona correctamente y que presenta índices de error significativamente elevados. Los clientes de esta carga de trabajo podrían dejar de realizar llamadas a sus servicios. Tal vez hayan activado un interruptor o hayan seguido su plan de recuperación de desastres para usar el servicio en otra región. Si solo estuviéramos midiendo las respuestas con errores, la disponibilidad de la carga de trabajo podría aumentar durante la avería, pero no porque la afectación mejorara o desapareciera, sino porque los clientes dejarían de usarla.

Latencia

Por último, también es importante medir la latencia de las unidades de trabajo de procesamiento dentro de la carga de trabajo. Parte de la definición de disponibilidad implica realizar el trabajo dentro de un SLA establecido. Si devolver una respuesta tarda más tiempo que el tiempo de espera del cliente, el cliente piensa que la solicitud ha fallado y que la carga de trabajo no está disponible. Sin embargo, en el servidor, puede parecer que la solicitud se ha procesado correctamente.

La medición de la latencia proporciona otra perspectiva con la que evaluar la disponibilidad. Los percentiles y la media recortada son buenas estadísticas para esta medición. Por lo general, se miden en el percentil 50 (P50 y MR50) y en el percentil 99 (P99 y MR99). La latencia debe medirse con valores controlados a la hora de representar la experiencia del cliente, así como con métricas del servidor. Cada vez que el promedio de algún percentil de latencia (como P99 o MR99,9) supere el SLA objetivo, se puede tener en cuenta ese tiempo de inactividad, lo que contribuye al cálculo del tiempo de inactividad anual.

Diseño de sistemas distribuidos de alta disponibilidad en AWS

Las secciones anteriores han tratado principalmente la disponibilidad teórica de las cargas de trabajo y lo que pueden lograr. Son un conjunto importante de conceptos que hay que tener en cuenta al crear sistemas distribuidos. Le orientarán a la hora de seleccionar las dependencias e implementar la redundancia.

También analizamos la MTTD relación entre MTTR y MTBF la disponibilidad. Esta sección presentará una guía práctica basada en la teoría anterior. En resumen, las cargas de trabajo de ingeniería para una alta disponibilidad tienen MTTR como objetivo aumentar MTBF y disminuir laMTTD.

Si bien lo ideal sería eliminar todos los errores, este objetivo no es nada realista. En sistemas distribuidos de gran tamaño con numerosas dependencias, se producirán errores. «Todo falla todo el tiempo» (véase Werner Vogels, Amazon.comCTO, 10 Lessons from 10 Years of Amazon Web Services) y «no se puede legislar contra el fracaso [así que] céntrese en la detección y la respuesta rápidas». (véase Chris Pinkham, miembro fundador del EC2 equipo de Amazon, ARC335Designing for failure: Architecting resilient systems on) AWS

Esto significa que, con frecuencia, el usuario no tiene control sobre si se produce un error. Lo que sí puede controlar es la rapidez con la que se detecta el error y se hace algo al respecto. Por lo tanto, si bien el aumento MTBF sigue siendo un componente importante de la alta disponibilidad, los cambios más importantes que los clientes tienen bajo su control son reducir MTTD y. MTTR

Temas

- Reduciendo MTTD
- Reduciendo MTTR
- Aumentando MTBF

Reduciendo MTTD

Reducir el MTTD número de fallos significa descubrirlos lo más rápido posible. La reducción MTTD se basa en la observabilidad o en la forma en que se ha instrumentado la carga de trabajo para comprender su estado. Los clientes deben supervisar sus métricas de experiencia de cliente en

los subsistemas críticos de sus cargas de trabajo para identificar de forma proactiva cuándo se produce un problema (consulte el apéndice 1) MTTD y las métricas MTTR críticas para obtener más información sobre estas métricas.). Los clientes pueden usar Amazon CloudWatch Synthetics para crear canarios que monitoreen sus consolas APIs y las suyas para medir de forma proactiva la experiencia del usuario. Existen otros mecanismos de comprobación de estado que se pueden utilizar para minimizarlaMTTD, como las comprobaciones de estado de Elastic Load Balancing (ELB), las comprobaciones de estado de Amazon Route 53 y más. (Consulte Amazon Builders' Library: implementación de las comprobaciones de estado).

Los mecanismos de supervisión también deben poder detectar errores parciales tanto en el sistema en su conjunto como en los subsistemas individuales. Sus métricas de disponibilidad, errores y latencia deben usar la dimensionalidad de los límites de aislamiento de fallas como dimensiones CloudWatch métricas. Por ejemplo, considere una EC2 instancia única que forma parte de una arquitectura basada en celdas, en la AZ use1-az1, en la región us-east-1, que forma parte de la actualización API de la carga de trabajo que forma parte de su subsistema de plano de control. Cuando el servidor introduce sus métricas, puede usar su identificador de instancia, la AZ, la región, el nombre y el nombre del subsistema como dimensiones. API Esto permite al usuario tener observabilidad y configurar alarmas en cada una de estas dimensiones para detectar errores.

Reduciendo MTTR

Una vez detectada una avería, el resto del MTTR tiempo se destinará a la reparación real o a la mitigación del impacto. Para reparar o mitigar un error, hay que saber qué es lo que está mal. Hay dos grupos clave de métricas que proporcionan información durante esta fase: 1/las métricas de evaluación del impacto y 2/las métricas del estado operativo. El primer grupo indica el alcance del impacto durante un error y mide el volumen o el porcentaje de clientes, recursos o cargas de trabajo afectados. El segundo grupo ayuda a identificar por qué se produce un impacto. Una vez identificado el motivo, los operadores y sistemas de automatización pueden responder al error y solucionarlo. Consulte el apéndice 1 MTTD y las métricas MTTR críticas para obtener más información sobre estas métricas.

> **Regla 9**
>
> La observabilidad y la instrumentación son fundamentales para reducir MTTD y. MTTR

Cómo sortear los errores

El enfoque más rápido para mitigar el impacto es mediante subsistemas que responden rápido a los errores y los sortean. Este enfoque utiliza la redundancia para reducir MTTR al transferir rápidamente el trabajo de un subsistema defectuoso a uno de repuesto. La redundancia puede abarcar desde procesos de software hasta EC2 instancias, múltiples o múltiples AZs regiones.

Los subsistemas de repuesto pueden MTTR reducirla prácticamente a cero. El tiempo de recuperación es lo único que se necesita para redirigir el trabajo al equipo de reserva o de repuesto. Esto suele ocurrir con una latencia mínima y permite que el trabajo se complete dentro de lo definidoSLA, manteniendo la disponibilidad del sistema. Esto produce MTTRs que se experimenten como retrasos leves, quizás incluso imperceptibles, en lugar de períodos prolongados de falta de disponibilidad.

Por ejemplo, si su servicio utiliza EC2 instancias detrás de un Application Load Balancer ALB (), puede configurar las comprobaciones de estado en un intervalo de tan solo cinco segundos y requerir solo dos comprobaciones de estado fallidas antes de que un objetivo se marque como en mal estado. Esto significa que, en 10 segundos, se puede detectar un error y dejar de enviar tráfico al host en mal estado. En este caso, MTTR es prácticamente el mismo que el anterior, MTTD ya que tan pronto como se detecta el error, también se mitiga.

Esto es lo que intentan lograr las cargas de trabajo de alta disponibilidad o disponibilidad continua. Queremos sortear rápidamente los errores en la carga de trabajo mediante la detección rápida de los subsistemas con errores, los marcamos como tal, dejamos de enviarles tráfico y, en su lugar, lo enviamos a un subsistema redundante.

Tenga en cuenta que el uso de este tipo de mecanismos que responden rápido a los errores hace que la carga de trabajo sea muy sensible a los errores transitorios. En el ejemplo que mostramos, asegúrese de que las comprobaciones de estado del equilibrador de carga se realicen de forma superficial o de manera dinámica y local para solamente la instancia, y de que las pruebas no se lleven a cabo para dependencias y flujos de trabajo (a menudo denominadas comprobaciones de estado exhaustivas). Esto ayudará a evitar la sustitución innecesaria de instancias durante errores transitorios que afecten a la carga de trabajo.

La observabilidad y la capacidad de detectar errores en los subsistemas son fundamentales para evitarlos de manera satisfactoria. Hay que conocer el alcance del impacto para que los recursos afectados puedan marcarse como en mal estado o con errores, y que queden fuera de servicio para poder sortearlos. Por ejemplo, si una única AZ sufre una interrupción parcial del servicio, su

instrumentación deberá poder identificar que existe un problema localizado en la AZ para poder distribuir todos los recursos de esa AZ hasta que se recupere el servicio.

Ser capaz de sortear un error también puede requerir herramientas adicionales en función del entorno. Si utilizamos el ejemplo anterior con EC2 instancias detrás de una zona de ALB disponibilidad, imagine que las instancias de una zona de disponibilidad podrían estar pasando las comprobaciones de estado locales, pero que un fallo de zona de disponibilidad aislado hace que no puedan conectarse a su base de datos en una zona de disponibilidad diferente. En este caso, las comprobaciones de estado del equilibrio de carga no dejarán esas instancias fuera de servicio. Se necesitaría un mecanismo automatizado diferente para eliminar la AZ del equilibrador de carga o forzar que las instancias no superen las comprobaciones de estado, lo que depende de haber identificado que el alcance del impacto es una AZ. En el caso de las cargas de trabajo que no utilizan un equilibrador de carga, se necesitaría un método similar para evitar que los recursos de una AZ específica acepten unidades de trabajo o eliminen completamente la capacidad de la zona.

En algunos casos, el traslado del trabajo a un subsistema redundante no se puede automatizar, como la conmutación por error de una base de datos principal a una secundaria, en la que la tecnología no elige a su propio líder. Este es un escenario común en las arquitecturas de varias regiones de AWS. Dado que estos tipos de conmutación por error requieren cierto tiempo de inactividad, no se pueden revertir de forma inmediata y dejan la carga de trabajo sin redundancia durante un período de tiempo, es importante contar con una persona que participe en el proceso de toma de decisiones.

Las cargas de trabajo que pueden adoptar un modelo de coherencia menos estricto pueden acortarse si utilizan la automatización de la conmutación MTTRs por error multirregional para evitar los errores. Características como la replicación entre regiones de Amazon S3 o las tablas globales de Amazon DynamoDB brindan prestaciones de varias regiones a través de una replicación eventualmente coherente. Además, usar un modelo de consistencia relajada es beneficioso si tenemos en cuenta el teorema. CAP Cuando se producen errores de red que afectan a la conectividad con los subsistemas activos, si la carga de trabajo elige la disponibilidad en lugar de la coherencia, puede seguir proporcionando respuestas sin errores, lo que supone otra forma de sortear los errores.

Los errores pueden sortearse con dos estrategias diferentes. La primera estrategia consiste en implementar la estabilidad estática mediante el aprovisionamiento previo de recursos suficientes para gestionar toda la carga del subsistema con errores. Puede ser una EC2 instancia única o puede ser una capacidad completa de AZ. Si se intenta aprovisionar nuevos recursos durante un error,

se incrementa el plano de control de la ruta de recuperación MTTR y se añade una dependencia al mismo. Sin embargo, conlleva un costo adicional.

La segunda estrategia consiste en enrutar parte del tráfico del subsistema con errores a otros y desbordar la carga con el exceso de tráfico que no pueda ser gestionado por la capacidad restante. Durante este período de degradación, se pueden escalar verticalmente nuevos recursos para sustituir la capacidad fallida. Este enfoque es más prolongado MTTR y crea una dependencia de un plano de control, pero cuesta menos si se trata de capacidad de reserva y sobrante.

Vuelta a un estado válido conocido

Otro enfoque habitual de mitigación durante la reparación consiste en devolver la carga de trabajo a un estado válido conocido anterior. Si un cambio reciente pudo haber provocado el error, revertir ese cambio es una forma de volver al estado anterior.

El error también podría deberse a condiciones transitorias, en cuyo caso, reiniciar la carga de trabajo podría mitigar el impacto. Vamos a analizar ambos escenarios.

Durante una implementación, se minimiza MTTD y MTTR se basa en la observabilidad y la automatización. El proceso de implementación debe vigilar continuamente la carga de trabajo para detectar la introducción de un aumento en los índices de error, un aumento de la latencia o la aparición de anomalías. Una vez se detectan estos problemas, el proceso de implementación debería interrumpirse.

Existen varias estrategias de implementación, como las implementaciones locales, las implementaciones azules/verdes o las implementaciones continuas. Cada una de ellas puede utilizar un mecanismo diferente para volver a un estado válido conocido. Puede restaurar automáticamente el estado anterior, devolver el tráfico al entorno azul o requerir una intervención manual.

CloudFormation ofrece la capacidad de revertirse automáticamente como parte de sus operaciones de creación y actualización de pilas, al igual que lo hace. AWS CodeDeploy CodeDeploy también es compatible con despliegues azul/verde y continuos.

Para aprovechar estas capacidades y minimizar las suyasMTTR, considere la posibilidad de automatizar todas las implementaciones de código e infraestructura a través de estos servicios. En los casos en los que no pueda utilizar estos servicios, considere la posibilidad de implementar el patrón Saga AWS Step Functions para revertir las implementaciones fallidas.

A la hora de plantearse el reinicio, existen varios enfoques diferentes. Estos van desde reiniciar un servidor (la tarea más larga) hasta reiniciar un subproceso (la tarea más corta). Esta tabla define

algunos de los métodos de reinicio y los tiempos aproximados para completarlos (representativos de diferencias de órdenes de magnitud; no son exactos).

Mecanismo de recuperación de errores	Estimación MTTR
Iniciar y configurar un servidor virtual nuevo	15 minutos
Reimplementar el software	10 minutos
Reiniciar el servidor	5 minutos
Reiniciar o iniciar el contenedor	2 segundos
Invocar una nueva función sin servidor	100 ms
Reiniciar el proceso	10 ms
Reiniciar el subproceso	10 µs

Si revisamos la tabla, podemos ver algunos beneficios evidentes del uso MTTR de contenedores y funciones sin servidor (por ejemplo AWS Lambda). MTTRSon órdenes de magnitud más rápidas que reiniciar una máquina virtual o lanzar una nueva. Sin embargo, utilizar el aislamiento de errores mediante la modularidad del software también conlleva una serie de ventajas. Si puede contener un error en un solo proceso o subproceso, recuperarse de ese error es mucho más rápido que reiniciar un contenedor o un servidor.

Como enfoque general para la recuperación, puede ir de abajo a arriba: 1/Reiniciar un componente, 2/Reiniciar el sistema completo, 3/Recrear la imagen/reimplementar y 4/Reemplazar. Sin embargo, una vez completado el proceso de reinicio del sistema completo, la solución al error suele ser más rápida (normalmente tarda entre 3 y 4 minutos como máximo). Por lo tanto, para mitigar el impacto de la forma más rápida posible tras un intento de reinicio, sortee el error y, en segundo plano, continúe con el proceso de recuperación para recuperar la capacidad de la carga de trabajo.

> **Regla 10**
>
> Concéntrese en la mitigación del impacto, no en la solución del problema. Recupere el funcionamiento normal por la vía más rápida.

Diagnóstico de errores

Parte del proceso de reparación después de la detección es el período de diagnóstico. Es el período de tiempo en el que los operadores tratan de determinar qué es lo que está mal. Este proceso puede implicar consultar registros, revisar métricas del estado operativo o iniciar sesión en los hosts para solucionar problemas. Todas estas acciones requieren tiempo, por lo que crear herramientas y manuales para agilizarlas también puede ayudar a reducirlas. MTTR

Manuales de procedimientos y automatización

Del mismo modo, después de determinar qué es lo que está mal y qué medidas tomar para reparar la carga de trabajo, los operadores suelen tener que realizar una serie de pasos para completar el proceso. Por ejemplo, después de un error, la forma más rápida de reparar la carga de trabajo puede ser reiniciarla, lo que puede implicar varios pasos ordenados. El uso de un manual de procedimientos que automatice estos pasos o proporcione instrucciones específicas al operador agilizará el proceso y ayudará a reducir el riesgo de que se tomen medidas involuntarias.

Aumentando MTBF

El último componente para mejorar la disponibilidad es aumentar laMTBF. Esto puede aplicarse tanto al software como a los AWS servicios utilizados para ejecutarlo.

Incrementar el sistema distribuido MTBF

Una forma de aumentarlo MTBF es reducir los defectos en el software. Puede hacer esto de varias formas. Los clientes pueden usar herramientas como Amazon CodeGuru Reviewer para encontrar y corregir errores comunes. También se deben realizar revisiones exhaustivas por pares del código, pruebas unitarias, pruebas de integración, pruebas de regresión y pruebas de carga del software antes de implementarlo en el entorno de producción. Aumentar la cobertura del código en las pruebas ayudará a garantizar que se prueben incluso las rutas de ejecución de código menos comunes.

Implementar cambios más pequeños también puede ayudar a evitar resultados inesperados al reducir la complejidad del cambio. Cada actividad brinda la oportunidad de identificar y corregir defectos antes de que puedan producirse.

Otro enfoque para prevenir los errores es realizar pruebas periódicas. La implementación de un programa de ingeniería del caos puede ayudar a comprobar si una carga de trabajo falla, validar los procedimientos de recuperación y ayudar a localizar y corregir los modos de error antes de que se

produzcan en el entorno de producción. Los clientes pueden usar AWS Fault Injection Simulator junto con otras herramientas para experimentos de ingeniería del caos.

La tolerancia a errores es otra forma de evitar errores en un sistema distribuido. Los módulos que responden rápido a los errores, los reintentos con fluctuación y retroceso exponenciales, las transacciones y la idempotencia son técnicas que ayudan a hacer que las cargas de trabajo sean tolerantes a los errores.

Las transacciones son un grupo de operaciones que se rigen por las ACID propiedades. Se definen de la siguiente manera:

- Atomicidad: o bien se producen todas las acciones o no ocurrirá ninguna de ellas.
- Coherencia: cada transacción deja la carga de trabajo en un estado válido.
- Aislamiento: las transacciones realizadas simultáneamente dejan la carga de trabajo en el mismo estado que si se hubieran realizado de forma secuencial.
- Durabilidad: una vez que se confirma una transacción, se conservan todos sus efectos, incluso en el caso de que se produzca un error en la carga de trabajo.

Los reintentos con fluctuación y retroceso exponenciales permiten superar los errores transitorios causados por errores Heisenbug, sobrecargas u otras condiciones. Cuando las transacciones son idempotentes, se pueden reintentar varias veces sin efectos secundarios.

Si tuviéramos en cuenta el efecto de un error Heisenbug en una configuración de hardware tolerante a errores, no nos preocuparía en absoluto, ya que la probabilidad de que el error Heisenbug aparezca tanto en el subsistema principal como en el redundante es infinitesimalmente pequeña. (Lea "Why Do Computers Stop and What Can Be Done About It?"; informe técnico 85.7 de Tandem Computers; junio de 1985). En los sistemas distribuidos, queremos lograr los mismos resultados con nuestro software.

Cuando se invoca un error Heisenbug, es imprescindible que el software detecte rápidamente un funcionamiento incorrecto y falle para volver a intentar la operación. Esto se logra mediante la programación defensiva y la validación de los datos de entrada, los resultados intermedios y los datos de salida. Además, los procesos están aislados y no comparten ningún estado con otros procesos.

Este enfoque modular garantiza que el alcance del impacto durante un error esté limitado. Los procesos fallan de forma independiente. Cuando un proceso falla, el software debe usar pares de procesos para reintentar el trabajo, lo que significa que un nuevo proceso puede asumir el trabajo

de un proceso con errores. Para mantener la fiabilidad e integridad de la carga de trabajo, cada operación debe tratarse como una ACID transacción.

Esto permite que un proceso falle sin dañar el estado de la carga de trabajo al abortar la transacción y revertir los cambios realizados. Así, el proceso de recuperación vuelve a intentar la transacción desde un estado válido conocido y se reinicia con elegancia. De esta forma, el software puede ser tolerante a los errores Heisenbug.

Sin embargo, no trate que el software sea tolerante a los errores Bohrbug. Estos defectos deben detectarse y eliminarse antes de que la carga de trabajo entre en el entorno de producción, ya que ningún nivel de redundancia logrará el resultado correcto. (Lea "Why Do Computers Stop and What Can Be Done About It?"; informe técnico 85.7 de Tandem Computers; junio de 1985).

La última forma de aumentarla MTBF es reducir el alcance del impacto provocado por un fallo. Usar el aislamiento de errores mediante modularización para crear contenedores de errores es una manera básica de hacerlo, como se describió anteriormente en Tolerancia a errores y aislamiento de errores. La reducción de la tasa de fallas mejora la disponibilidad. AWS utiliza técnicas como la división de los servicios en planos de control y planos de datos, la independencia de la zona de disponibilidad (AZI), el aislamiento regional, las arquitecturas basadas en celdas y la fragmentación aleatoria para aislar las fallas. Estos también son patrones que los clientes también pueden utilizar. AWS

Analicemos como ejemplo un escenario en el que una carga de trabajo ubica a los clientes en diferentes contenedores de errores de una infraestructura que presta servicio a como máximo el 5 % de los clientes totales. En uno de estos contenedores de errores se produce un evento que aumenta la latencia una vez transcurrido el tiempo de espera del cliente en un 10 % de las solicitudes. Durante este evento, para el 95 % de los clientes, el servicio estuvo disponible al 100 %. En el caso del 5 % restante, el servicio parecía estar disponible al 90 %. Esto se traduce en una disponibilidad de 1 − (5 % de clientes× 10 % de sus solicitudes) = 99,5 % en lugar del 10 % de las solicitudes con errores para el 100 % de los clientes (lo que se traduce en una disponibilidad del 90 %).

> Regla 11
>
> El aislamiento de fallas reduce el alcance del impacto y aumenta la carga MTBF de trabajo al reducir la tasa general de fallas.

Aumento de la dependencia MTBF

El primer método para aumentar la AWS dependencia MTBF es mediante el uso del aislamiento de errores. Muchos AWS servicios ofrecen un nivel de aislamiento en la AZ, lo que significa que una falla en una AZ no afecta al servicio en otra AZ.

El uso de EC2 instancias redundantes en varias instancias AZs aumenta la disponibilidad del subsistema. AZIproporciona una capacidad de ahorro dentro de una sola región, lo que le permite aumentar la disponibilidad de los servicios. AZI

Sin embargo, no todos los AWS servicios funcionan a nivel de zona de disponibilidad. Muchos otros ofrecen aislamiento regional. En este caso, cuando la disponibilidad diseñada del servicio regional no respalde la disponibilidad general requerida para la carga de trabajo, podría plantearse aplicar un enfoque de varias regiones. Cada región ofrece una instanciación aislada del servicio, lo que equivale a reservas.

Existen varios servicios que facilitan la creación de un servicio de varias regiones. Por ejemplo:

- Base de datos global de Amazon Aurora
- Tablas globales de Amazon DynamoDB
- Amazon ElastiCache (RedisOSS): almacén de datos global
- AWS Acelerador global
- Replicación entre regiones de Amazon S3
- Controlador de recuperación de aplicaciones de Amazon Route 53

Este documento no profundiza en las estrategias para crear cargas de trabajo de varias regiones, por lo que debe sopesar los beneficios en cuanto a disponibilidad de las arquitecturas de varias regiones y el costo, la complejidad y las prácticas operativas adicionales y necesarias para satisfacer los objetivos de disponibilidad deseados.

El siguiente método para aumentar la dependencia MTBF consiste en diseñar la carga de trabajo para que sea estable desde el punto de vista estático. Supongamos que, por ejemplo, tiene una carga de trabajo que proporciona información sobre un producto. Cuando sus clientes solicitan un producto, el servicio realiza una solicitud a un servicio de metadatos externo que recupera los detalles del producto. Luego, su carga de trabajo devuelve toda esa información al usuario.

Sin embargo, si el servicio de metadatos no está disponible, las solicitudes realizadas por los clientes fallarán. En cambio, puede extraer o enviar los metadatos de forma local y asíncrona a su servicio

para utilizarlos en las respuestas a las solicitudes. Así se elimina la llamada síncrona al servicio de metadatos desde la ruta crítica.

Además, dado que su servicio sigue estando disponible aunque el servicio de metadatos no lo esté, puede eliminarlo como una dependencia en el cálculo de la disponibilidad. Este ejemplo se basa en la suposición de que los metadatos no cambian con frecuencia y que es mejor publicarlos obsoletos que incurrir en errores de solicitud. Otro ejemplo similar es el de serve-stale, DNS que permite conservar los datos en la memoria caché después de su TTL vencimiento y utilizarlos como respuestas cuando no se dispone fácilmente de una respuesta actualizada.

El último método para aumentar la dependencia MTBF consiste en reducir el alcance del impacto de un error. Como se mencionó anteriormente, un error no es un evento binario, sino que existen grados de error. Este es el efecto de la modularización; el error se limita a las solicitudes o a los usuarios a los que presta servicio ese contenedor.

Esto se traduce en menos errores durante un evento, lo que, en última instancia, aumenta la disponibilidad general de la carga de trabajo al limitar el alcance del impacto.

Cómo reducir las fuentes de impacto más habituales

En 1985, Jim Gray descubrió durante un estudio en Tandem Computers que los errores se debían principalmente a dos factores: el software y las operaciones. (Lea “Why Do Computers Stop and What Can Be Done About It?”; informe técnico 85.7 de Tandem Computers; junio de 1985). 36 años después, esto sigue siendo cierto. A pesar de los avances tecnológicos, no existe una solución fácil para estos problemas y las principales fuentes de error no han cambiado. Al principio de esta sección se abordaron los errores en el software, por lo que ahora vamos a centrarnos en las operaciones y en reducir la frecuencia con la que ocurren estos errores.

Comparativa entre la estabilidad y las características

Si volvemos a consultar el gráfico del índice de errores del software y el hardware de la sección the section called “Disponibilidad de los sistemas distribuidos”, podemos observar que en cada versión de software se añaden nuevos defectos. Esto significa que cualquier cambio en la carga de trabajo conlleva un mayor riesgo de error. Estos cambios suelen consistir en nuevas funciones, por ejemplo, lo que constituye un corolario. Las cargas de trabajo de mayor disponibilidad favorecerán la estabilidad frente a las nuevas características. Por lo tanto, una de las formas más sencillas de mejorar la disponibilidad es implementarlas con menor frecuencia u ofrecer menos. Las cargas de trabajo que se implementan con más frecuencia tendrán por naturaleza una disponibilidad menor que

las que no lo hacen. Sin embargo, las cargas de trabajo que no incorporan nuevas características no satisfacen la demanda de los clientes y pueden perder utilidad con el tiempo.

¿Cómo podemos seguir innovando y lanzando características de forma segura? La estandarización es la respuesta. ¿Cuál es la forma correcta de implementación? ¿Cómo se solicitan las implementaciones? ¿Qué estándares hay para las pruebas? ¿Cuánto tiempo se espera entre una etapa y otra? ¿Cubren sus pruebas unitarias suficiente código de software? Estas son preguntas a las que la estandarización responderá y evitará problemas causados por factores como la falta de pruebas de cargas, la omisión de las etapas de implementación o la implementación demasiado rápida en demasiados hosts.

La forma de implementar la estandarización es mediante la automatización. Reduce la posibilidad de que se cometan errores humanos y permite a los ordenadores hacer aquello que se les da bien; es decir, hacer siempre lo mismo una y otra vez de la misma manera. La manera de combinar la estandarización y la automatización es establecer objetivos. Los objetivos incluyen evitar los cambios manuales, permitir el acceso al host únicamente mediante sistemas de autorización contingente, escribir pruebas de carga para cada uno de ellosAPI, etc. La excelencia operativa es una norma cultural que puede requerir cambios sustanciales. Establecer y hacer un seguimiento del rendimiento en relación con un objetivo ayuda a impulsar un cambio cultural que tendrá una gran repercusión en la disponibilidad de la carga de trabajo. El Pilar de excelencia operativa del marco de AWS Well-Architected brinda prácticas recomendadas integrales para la excelencia operativa.

Seguridad de los operadores

El otro factor importante que contribuye a los eventos operativos que provocan errores son las personas. Los seres humanos cometemos errores. Podemos usar credenciales incorrectas, introducir comandos incorrectos, pulsar Intro demasiado pronto o saltarnos un paso importante. Tomar medidas manuales de forma sistemática causa errores.

Una de las principales causas de los errores de los operadores son las interfaces de usuario confusas, poco intuitivas o incoherentes. Jim Gray también señaló en su estudio de 1985 que “las interfaces que solicitan información al operador o le piden que realice alguna función deben ser simples, coherentes y tolerantes a los errores del operador”. (Lea “Why Do Computers Stop and What Can Be Done About It?”; informe técnico 85.7 de Tandem Computers; junio de 1985). Esta idea sigue siendo cierta en la actualidad. En las últimas tres décadas, en el sector, se han dado numerosos ejemplos en los que una interfaz de usuario confusa o compleja, la falta de confirmación o instrucciones, o incluso un lenguaje humano poco amigable hicieron que un operador se equivocara.

Regla 12

Facilite que los operadores hagan lo correcto.

Prevención de sobrecargas

El último factor de impacto habitual son los clientes, los usuarios reales de la carga de trabajo. Las cargas de trabajo que funcionan bien suelen utilizarse mucho, pero a veces ese uso supera su capacidad de escalabilidad. Pueden ocurrir muchas cosas: los discos pueden llenarse, los grupos de subprocesos pueden agotarse, el ancho de banda de la red puede saturarse o se pueden alcanzar los límites de conexión a la base de datos.

No existe un método infalible para evitar estos problemas, pero la supervisión proactiva de la capacidad y la utilización a través de las métricas del estado operativo proporcionará alertas tempranas cuando se produzcan estos errores. Técnicas como el desbordamiento de carga, el uso de interruptores y reintentos con fluctuación y retroceso exponenciales pueden ayudar a minimizar el impacto y mejorar el índice de éxito, pero estas situaciones siguen siendo errores. El escalado automatizado basado en las métricas del estado operativo puede ayudar a reducir la frecuencia con la que se producen errores debidos a una sobrecarga, pero es posible que no pueda responder con la suficiente rapidez a los cambios en la utilización.

Si necesita garantizar la capacidad disponible de forma continua para los clientes, debe hacer concesiones en cuanto a la disponibilidad y el costo. Una forma de garantizar que la falta de capacidad no provoque una falta de disponibilidad consiste en proporcionar a cada cliente una cuota y garantizar que la capacidad de la carga de trabajo se adapte para cubrir el 100 % de las cuotas asignadas. Cuando los clientes superan su cuota, se exponen a limitaciones, pero no se produce ningún error ni la disponibilidad se ve afectada. También tendrá que realizar un seguimiento minucioso de su base de clientes y pronosticar la utilización futura para mantener suficiente capacidad aprovisionada. Esto garantizará que la carga de trabajo no se vea obligada a fallar debido al consumo excesivo por parte de los clientes.

- Amazon Builders' Library: uso del desbordamiento de carga para evitar la sobrecarga
- Amazon Builders' Library: Fairness in multi-tenant systems

Tomemos como ejemplo una carga de trabajo que proporciona un servicio de almacenamiento. Cada servidor de la carga de trabajo puede admitir 100 descargas por segundo, a los clientes se les proporciona una cuota de 200 descargas por segundo y hay 500 clientes. Para poder soportar este

volumen de clientes, el servicio debe ofrecer una capacidad de 100 000 descargas por segundo, lo que requiere 1000 servidores. Si algún cliente supera su cuota, se verá expuesto a una limitación, lo que garantizará una capacidad suficiente para el resto de los clientes. Este es un ejemplo sencillo de cómo evitar una sobrecarga sin rechazar unidades de trabajo.

Conclusión

Hemos definido 12 reglas para la alta disponibilidad a lo largo de este documento.

- Regla 1: Los tres factores que se utilizan para mejorar la disponibilidad en los sistemas distribuidos son una menor frecuencia de errores (MTBF más largo), tiempos de detección de errores más cortos (MTTD más corto) y tiempos de reparación más cortos (MTTR más corto).
- Regla 2: La disponibilidad del software en su carga de trabajo es un factor importante de la disponibilidad general de la carga de trabajo y debe recibir la misma atención que otros componentes.
- Regla 3: Reducir las dependencias puede tener un impacto positivo en la disponibilidad.
- Regla 4: En general, seleccione las dependencias cuyos objetivos de disponibilidad sean iguales o superiores a los objetivos de su carga de trabajo.
- Regla 5: Utilice repuestos para incrementar la disponibilidad de las dependencias en una carga de trabajo.
- Regla 6: La rentabilidad del uso de repuestos tiene un límite superior. Utilice el menor número de repuestos necesario para lograr la disponibilidad requerida.
- Regla 7: No tenga dependencias en los planos de control en el plano de datos, especialmente durante una recuperación.
- Regla 8: Cuando sea posible, acople las dependencias de manera flexible para que la carga de trabajo pueda funcionar correctamente a pesar del deterioro de la dependencia.
- Regla 9: La observabilidad y la instrumentación son fundamentales para reducir los tiempos medios de detección y de recuperación (MTTD y MTTR, respectivamente).
- Regla 10: Concéntrese en la mitigación del impacto, no en la solución del problema. Recupere el funcionamiento normal por la vía más rápida.
- Regla 11: El aislamiento de errores reduce el alcance del impacto e incrementa el MTBF de la carga de trabajo mediante la reducción del índice general de errores.
- Regla 12: Facilite que los operadores hagan lo correcto.

La disponibilidad de la carga de trabajo se mejora mediante la reducción del MTTD y el MTTR, y con el aumento del MTBF. En resumen, hemos analizado las siguientes formas de mejorar la disponibilidad, que abarcan la tecnología, las personas y los procesos.

- MTTD

  - Reduzca el MTTD mediante la supervisión proactiva de las métricas de la experiencia del cliente.
  - Aproveche las comprobaciones de estado detalladas para una rápida conmutación por error.
- MTTR
  - Supervise las métricas del alcance del impacto y del estado operativo.
  - Reduzca el MTTR siguiendo los pasos 1/Reiniciar un componente, 2/Reiniciar el sistema completo, 3/Reinstalar imagen/Reimplementar y 4/Reemplazar.
  - Sortee los errores conociendo el alcance del impacto.
  - Utilice servicios que tengan tiempos de reinicio más rápidos, como contenedores y funciones sin servidor, en lugar de máquinas virtuales o hosts físicos.
  - Restaure automáticamente las implementaciones con errores siempre que sea posible.
  - Establezca manuales de procedimientos y herramientas operativas para las operaciones de diagnóstico y los procedimientos de reinicio.
- MTBF
  - Elimine los errores y defectos del software mediante rigurosas pruebas antes de su lanzamiento al entorno de producción.
  - Implemente la ingeniería del caos y la inyección de errores.
  - Utilice el volumen adecuado de capacidad de reserva en las dependencias para tolerar los errores.
  - Minimice el alcance del impacto durante los errores con contenedores de errores.
  - Implemente estándares para las implementaciones y los cambios.
  - Diseñe interfaces de operador sencillas, intuitivas, coherentes y bien documentadas.
  - Establezca objetivos en torno a la excelencia operativa.
  - Favorezca la estabilidad frente al lanzamiento de nuevas características cuando la disponibilidad sea una dimensión fundamental de la carga de trabajo.
  - Implemente cuotas con limitaciones o desborde la carga, o bien haga ambas cosas, para evitar sobrecargas.

Recuerde que nunca lograremos evitar por completo los errores. Céntrese en los diseños de software con el mejor aislamiento posible de errores, que limite el alcance y la magnitud del impacto, idealmente manteniendo ese impacto por debajo de los umbrales de tiempo de inactividad. Invierta también en herramientas muy rápidas y muy fiables de detección y mitigación. Los sistemas

distribuidos modernos aún deben considerar los errores como algo inevitable y diseñarse a todos los niveles para ofrecer una alta disponibilidad.

Apéndice 1: Métricas fundamentales del MTTD y el MTTR

Lo siguiente es un marco para la estandarización de la instrumentación y la observabilidad que puede ayudar a reducir los tiempos medios de detección y de recuperación (MTTD y MTTR, respectivamente). durante un evento.

Métricas de la experiencia del cliente: reflejan que un servicio responde y está disponible para atender las solicitudes de los clientes. Por ejemplo, la latencia del plano de control. Estas métricas miden el índice de errores, la disponibilidad, la latencia, el volumen y el índice de limitaciones.

Métricas de evaluación del impacto: proporcionan información sobre el alcance del impacto durante los eventos. Por ejemplo, la cantidad o el porcentaje de clientes afectados por un evento del plano de datos. Mide la cantidad o el porcentaje de cosas afectadas.

Métricas del estado operativo: reflejan que un servicio responde y está disponible para atender las solicitudes de los clientes, pero se centran en los subsistemas y los recursos habituales de la infraestructura. Por ejemplo, el porcentaje de uso de la CPU de la flota de EC2. Estas métricas deben medir la utilización, la capacidad, el rendimiento, el índice de errores, la disponibilidad y la latencia.

Colaboradores

Los colaboradores de este documento son:

- Michael Haken, arquitecto principal de soluciones, Amazon Web Services

Documentación adicional

Para obtener información adicional, consulte los siguientes recursos:

- Pilar de fiabilidad del marco Well-Architected Framework
- Pilar de excelencia operativa del marco Well-Architected Framework
- Amazon Builders' Library: Asegurar la seguridad en las restauraciones durante las implementaciones
- Amazon Builders' Library: Beyond five 9s: Lessons from our highest available data planes
- Amazon Builders' Library: “Automatización de implementaciones seguras y sin intervención”
- Amazon Builders' Library: Architecting and operating resilient serverless systems at scale
- Amazon Builders' Library: Amazon's approach to high-availability deployment
- Amazon Builders' Library: Amazon's approach to building resilient services
- Amazon Builders' Library: Amazon’s approach to failing successfully
- Centro de arquitectura de AWS

Historial de documentos

Para recibir notificaciones sobre las actualizaciones de este documento técnico, suscríbase a la fuente RSS.

Cambio	Descripción	Fecha
Publicación inicial	Documento técnico publicado por primera vez.	12 de noviembre de 2021

> **Note**
>
> Si quiere suscribirse a las actualizaciones en formato RSS, debe tener un complemento de RSS habilitado en el navegador que esté utilizando.

Avisos

Es responsabilidad de los clientes realizar su propia evaluación independiente de la información que contiene este documento. El presente documento: (a) tiene solo fines informativos, (b) representa las ofertas y prácticas actuales de los productos de AWS, que están sujetas a cambios sin previo aviso, y (c) no supone ningún compromiso ni garantía por parte de AWS y sus filiales, proveedores o licenciantes. Los productos o servicios de AWS se proporcionan “tal cual”, sin garantías, afirmaciones ni condiciones de ningún tipo, ya sean expresas o implícitas. Las responsabilidades y obligaciones de AWS con respecto a sus clientes se controlan mediante los acuerdos de AWS y este documento no forma parte ni modifica ningún acuerdo entre AWS y sus clientes.

Glosario de AWS

Para ver la terminología más reciente de AWS, consulte el Glosario de AWS en la Referencia de Glosario de AWS.

Las traducciones son generadas a través de traducción automática. En caso de conflicto entre la traducción y la version original de inglés, prevalecerá la version en inglés.