Whitepaper da AWS

Disponibilidade e muito mais: entendendo e melhorando a resiliência de sistemas distribuídos no AWS

Disponibilidade e muito mais: entendendo e melhorando a resiliência de sistemas distribuídos no AWS: Whitepaper da AWS

Table of Contents

Resumo e introdução i
Introdução 1
Noções básicas da disponibilidade 2
Disponibilidade do sistema distribuído 4
Tipos de falhas em sistemas distribuídos 6
Disponibilidade com dependências 6
Disponibilidade com redundância 8
Teorema CAP 13
Tolerância a falhas e isolamento de falhas 14
Medindo a disponibilidade 17
Taxa sobre o sucesso das solicitações nos lados do servidor e do cliente 17
Tempo de inatividade anual 20
Latência 21
Projetando sistemas distribuídos de alta disponibilidade no AWS 22
Redução do MTTD 22
Redução do MTTD 23
Rota para contornar falha 23
Retorne a um bom estado conhecido 26
Diagnóstico de falhas 27
Runbooks e automação 28
Aumento do MTBF 28
Aumentando o MTBF do sistema distribuído 28
Aumento da dependência MTBF 30
Reduzindo fontes comuns de impacto 32
Conclusão 35
Apêndice 1 — Métricas críticas de MTTD e MTTR 37
Colaboradores 38
Outras fontes de leitura 39
Histórico do documento 40
Avisos 41
Glossário do AWS 42

Disponibilidade e muito mais: entendendo e melhorando a resiliência de sistemas distribuídos no AWS

Data de publicação: 12 de novembro de 2021 (Histórico do documento)

Atualmente, as empresas operam sistemas distribuídos complexos na nuvem e no local. Eles querem que essas workloads sejam resilientes para atender seus clientes e alcançar seus resultados comerciais. Este paper descreve um entendimento comum da disponibilidade como medida de resiliência, estabelece regras para criar workloads altamente disponíveis e oferece orientação sobre como melhorar a disponibilidade da workload.

Introdução

O que significa criar uma workload altamente disponível? Como você mede a disponibilidade? O que posso fazer para aumentar a disponibilidade da minha workload? Este documento ajudará você a responder a esses tipos de perguntas. É dividido em três seções principais. A primeira seção, Entendendo a disponibilidade, é em grande parte teórica. Ele estabelece um entendimento comum da definição de disponibilidade e dos fatores que a impactam. A segunda seção, Medindo a disponibilidade, fornece orientação sobre como medir empiricamente a disponibilidade da sua workload. A terceira seção, Projetando sistemas distribuídos altamente disponíveis no AWS, é uma aplicação prática das ideias apresentadas na primeira seção. Além disso, ao longo dessas seções, este paper identificará regras para criar workloads resilientes. Este documento tem como objetivo apoiar a orientação e as melhores práticas apresentadas no AWS Well-Architected Reliability Pillar.

Ao longo deste paper, você encontrará muita matemática algébrica. As principais conclusões são os conceitos que essa matemática apoia, não a matemática em si. Dito isso, também é a intenção deste paper apresentar um desafio. Ao operar workloads altamente disponíveis, você precisa ser capaz de provar, matematicamente, que o que você construiu está alcançando o que você pretendia. Mesmo os melhores designs baseados em boas intenções podem não alcançar consistentemente o resultado desejado. Isso significa que você precisa de mecanismos que meçam a eficácia da solução e, portanto, algum nível de matemática é necessário para criar e operar sistemas distribuídos resilientes e altamente disponíveis.

Noções básicas da disponibilidade

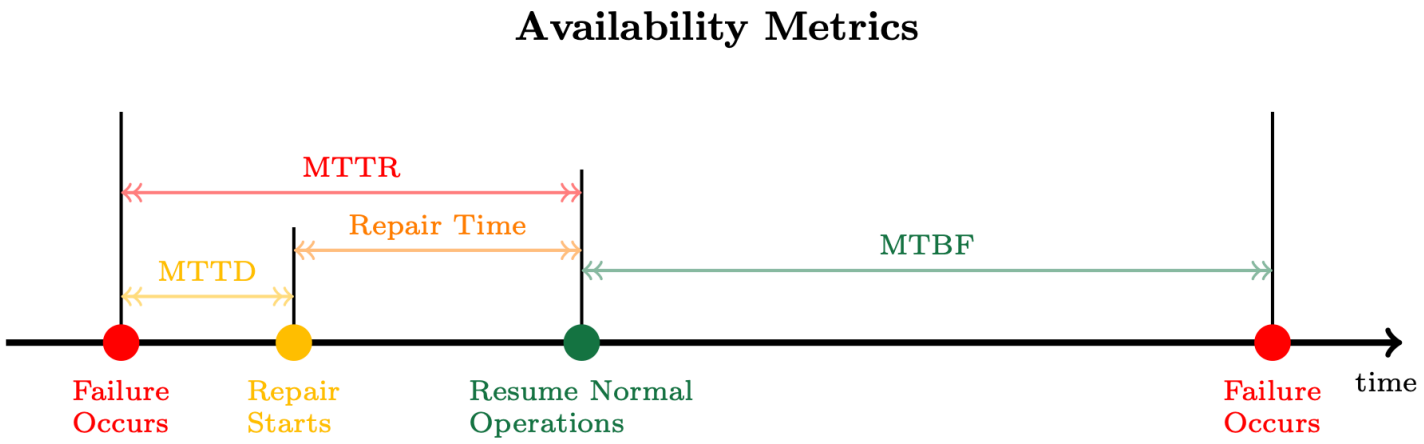
A disponibilidade é uma das principais formas de medir quantitativamente a resiliência. Definimos disponibilidade, A, como a porcentagem de tempo em que uma workload está disponível para uso. É uma proporção entre o “tempo de atividade” esperado (disponibilidade) e o tempo total medido (o “tempo de atividade” esperado mais o “tempo de inatividade” esperado).

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

Equação 1 - Disponibilidade

Para entender melhor essa fórmula, veremos como medir o tempo de atividade e o tempo de inatividade. Primeiro, queremos saber quanto tempo a workload durará sem falhas. Chamamos isso de tempo médio entre falhas (MTBF), o tempo médio entre o início da operação normal de uma workload e sua próxima falha. Então, queremos saber quanto tempo levará para se recuperar após a falha.

Chamamos isso de tempo médio de reparo (ou recuperação) (MTTR), um período em que a workload não está disponível enquanto o subsistema com defeito é reparado ou retornado ao serviço. Um período de tempo importante no MTTR é o tempo médio de detecção (MTTD), a quantidade de tempo entre a ocorrência de uma falha e o início das operações de reparo. O diagrama a seguir demonstra como todas essas métricas estão relacionadas.



A relação entre MTTD, MTTR e MTBF

Assim, podemos expressar disponibilidade, A, usando MTBF, quando a workload está alta, e MTTR, quando a workload está inativa.

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

Equação 2 - Relação entre MTBF e MTTR

E a probabilidade de a workload estar “inativa” (ou seja, não disponível) é a probabilidade de falha, F.

$$F = 1 - A$$

Equação 3 - Probabilidade de falha

Confiabilidade é a capacidade de uma workload fazer a coisa certa, quando solicitada, dentro do tempo de resposta especificado. É isso que a disponibilidade mede. Ter uma workload falhar com menos frequência (MTBF mais longo) ou ter um tempo de reparo mais curto (MTTR mais curto) melhora sua disponibilidade.

> Rule1
>
> Falhas menos frequentes (MTBF mais longo), tempos de detecção de falhas mais curtos (MTTD mais curto) e tempos de reparo mais curtos (MTTR mais curto) são os três fatores usados para melhorar a disponibilidade em sistemas distribuídos.

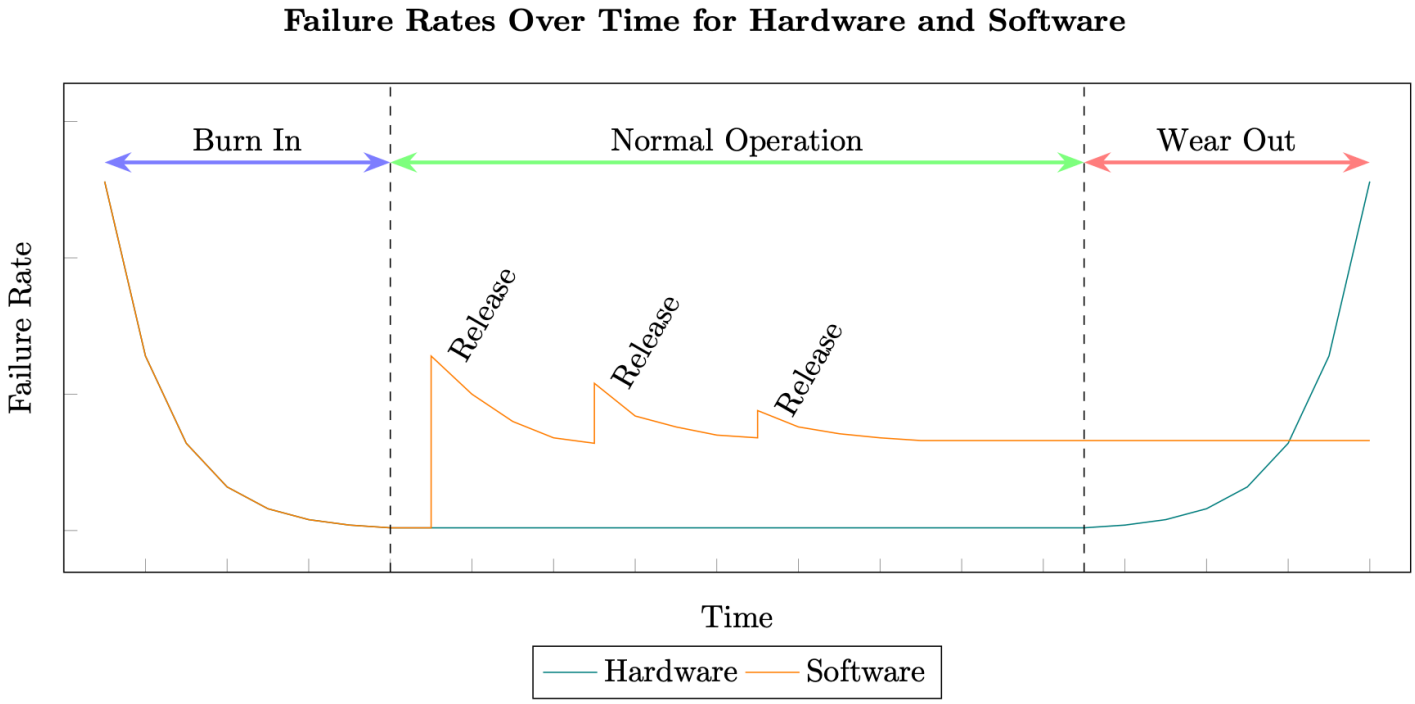
Tópicos

- Disponibilidade do sistema distribuído
- Disponibilidade com dependências
- Disponibilidade com redundância
- Teorema CAP
- Tolerância a falhas e isolamento de falhas

Disponibilidade do sistema distribuído

Os sistemas distribuídos são compostos por componentes de software e componentes de hardware. Alguns dos componentes do software podem ser, eles próprios, outro sistema distribuído. A disponibilidade dos componentes subjacentes de hardware e software afeta a disponibilidade resultante de sua workload.

O cálculo da disponibilidade usando MTBF e MTTR tem suas raízes nos sistemas de hardware. No entanto, os sistemas distribuídos falham por motivos muito diferentes dos de um hardware. Quando um fabricante pode calcular consistentemente o tempo médio antes que um componente de hardware se desgaste, o mesmo teste não pode ser aplicado aos componentes de software de um sistema distribuído. O hardware normalmente segue a curva “banhada” da taxa de falhas, enquanto o software segue uma curva escalonada produzida por defeitos adicionais que são introduzidos a cada nova versão (consulte Confiabilidade do software).



Taxas de falha de hardware e software

Além disso, o software em sistemas distribuídos normalmente muda a taxas exponencialmente maiores do que o hardware. Por exemplo, um disco rígido magnético padrão pode ter uma taxa média de falha anualizada (AFR) de 0,93%, o que, na prática, para um HDD, pode significar uma vida útil de pelo menos 3 a 5 anos antes de atingir o período de desgaste, potencialmente mais longa (consulte Dados e estatísticas do disco rígido Backblaze, 2020). O disco rígido não muda

materialmente durante essa vida útil, onde, em 3 a 5 anos, por exemplo, a Amazon pode implantar mais de 450 a 750 milhões de alterações em seus sistemas de software. (Consulte Amazon Builders' Library — Automatização de implantações seguras e sem intervenção.)

O hardware também está sujeito ao conceito de obsolescência planejada, ou seja, tem uma vida útil integrada e precisará ser substituído após um determinado período de tempo. (Veja A Grande Conspiração da Lâmpada.) O software, teoricamente, não está sujeito a essa restrição, não tem um período de desgaste e pode ser operado indefinidamente.

Tudo isso significa que os mesmos modelos de teste e previsão usados pelo hardware para gerar números MTBF e MTTR não se aplicam ao software. Houve centenas de tentativas de criar modelos para resolver esse problema desde a década de 1970, mas todas elas geralmente se enquadram em duas categorias: modelagem de previsão e modelagem de estimativa. (Consulte a lista de modelos de confiabilidade de software.)

Assim, o cálculo de um MTBF e MTTR prospectivos para sistemas distribuídos e, portanto, de uma disponibilidade prospectiva, sempre será derivado de algum tipo de previsão ou previsão. Eles podem ser gerados por meio de modelagem preditiva, simulação estocástica, análise histórica ou testes rigorosos, mas esses cálculos não garantem tempo de atividade ou inatividade.

Os motivos pelos quais um sistema distribuído falhou no passado podem nunca mais ocorrer. As razões pelas quais ele falhará no futuro provavelmente serão diferentes e possivelmente desconhecidas. Os mecanismos de recuperação necessários também podem ser diferentes para futuras falhas dos usados no passado e levar períodos de tempo significativamente diferentes.

Além disso, MTBF e MTTR são médias. Haverá alguma variação do valor médio em relação aos valores reais vistos (o desvio padrão, σ, mede essa variação). Assim, as workloads podem passar por um tempo menor ou maior entre falhas e tempos de recuperação no uso real da produção.

Dito isso, a disponibilidade dos componentes de software que compõem um sistema distribuído ainda é importante. O software pode falhar por vários motivos (discutidos mais na próxima seção) e afetar a disponibilidade da workload. Assim, para sistemas distribuídos de alta disponibilidade, o mesmo foco no cálculo, medição e melhoria da disponibilidade dos componentes de software deve ser dado ao hardware e aos subsistemas externos de software.

> **Rule2**
>
> A disponibilidade do software em sua workload é um fator importante da disponibilidade geral de sua workload e deve receber o mesmo foco que outros componentes.

É importante observar que, apesar de serem difíceis de prever o MTBF e o MTTR para sistemas distribuídos, eles ainda fornecem informações importantes sobre como melhorar a disponibilidade. Reduzir a frequência de falhas (MTBF mais alto) e diminuir o tempo de recuperação após a ocorrência da falha (MTTR mais baixo) levarão a uma maior disponibilidade empírica.

Tipos de falhas em sistemas distribuídos

Geralmente, há duas classes de bugs em sistemas distribuídos que afetam a disponibilidade, carinhosamente chamadas de Bohrbug e Heisenbug (consulte "Uma conversa com Bruce Lindsay", ACM Queue vol. 2, nº 8 – novembro de 2004.)

Um Bohrbug é um problema repetível de software funcional. Com a mesma entrada, o bug produzirá consistentemente a mesma saída incorreta (como o modelo determinístico do átomo de Bohr, que é sólido e facilmente detectado). Esses tipos de bugs são raros quando uma workload chega à produção.

Um Heisenbug é um bug transitório, o que significa que só ocorre em condições específicas e incomuns. Essas condições geralmente estão relacionadas a coisas como hardware (por exemplo, uma falha transitória do dispositivo ou especificidades da implementação de hardware, como tamanho do registro), otimizações do compilador e implementação da linguagem, condições de limite (por exemplo, temporariamente fora do armazenamento) ou condições de corrida (por exemplo, não usar um semáforo para operações com vários processos).

Os Heisenbugs compõem a maioria dos bugs em produção e são difíceis de encontrar porque são ilusórios e parecem mudar de comportamento ou desaparecer quando você tenta observá-los ou depurá-los. No entanto, se você reiniciar o programa, a operação com falha provavelmente será bem-sucedida porque o ambiente operacional é um pouco diferente, eliminando as condições que introduziram o Heisenbug.

Assim, a maioria das falhas na produção são transitórias e, quando a operação é repetida, é improvável que falhe novamente. Para serem resilientes, os sistemas distribuídos precisam ser tolerantes a falhas da Heisenbugs. Exploraremos como isso pode ser feito na seção Aumentando o MTBF do sistema distribuído.

Disponibilidade com dependências

Na seção anterior, mencionamos que hardware, software e potencialmente outros sistemas distribuídos são todos componentes de sua workload. Chamamos esses componentes de

dependências, as coisas das quais sua workload depende para fornecer sua funcionalidade. Existem dependências rígidas, que são aquelas coisas sem as quais sua workload não pode funcionar, e dependências flexíveis cuja indisponibilidade pode passar despercebida ou tolerada por algum período de tempo. Dependências rígidas têm um impacto direto na disponibilidade da sua workload.

Talvez queiramos tentar calcular a disponibilidade máxima teórica de uma workload. Esse é o produto da disponibilidade de todas as dependências, incluindo o próprio software (α_n é a disponibilidade de um único subsistema) porque cada uma deve estar operacional.

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

Equação 4 - Disponibilidade máxima teórica

Os números de disponibilidade usados nesses cálculos geralmente estão associados a itens como SLAs ou objetivos de nível de serviço (SLOs). Os SLAs definem o nível esperado de serviço que os clientes receberão, as métricas pelas quais o serviço é medido e as remediações ou penalidades (geralmente monetárias) caso os níveis de serviço não sejam alcançados.

Usando a fórmula acima, podemos concluir que, puramente matematicamente, uma workload não pode estar mais disponível do que qualquer uma de suas dependências. Mas, na realidade, o que normalmente vemos é que esse não é o caso. Uma workload criada usando duas ou três dependências com SLAs de 99,99% de disponibilidade ainda pode atingir 99,99% de disponibilidade sozinha, ou mais.

Isso ocorre porque, conforme descrevemos na seção anterior, esses números de disponibilidade são estimativas. Eles estimam ou preveem com que frequência uma falha ocorre e com que rapidez ela pode ser reparada. Eles não são garantia de tempo de inatividade. As dependências frequentemente excedem o SLA ou SLO de disponibilidade declarado.

As dependências também podem ter objetivos de disponibilidade interna mais altos em relação aos quais elas visam o desempenho do que os números fornecidos em SLAs públicos. Isso fornece um nível de mitigação de risco no cumprimento dos SLAs quando o desconhecido ou incognoscível acontece.

Por fim, sua workload pode ter dependências cujos SLAs não podem ser conhecidos ou não oferecem um SLA ou SLO. Por exemplo, o roteamento mundial da Internet é uma dependência comum para muitas workloads, mas é difícil saber quais provedores de serviços de Internet seu tráfego global está usando, se eles têm SLAs e se são consistentes entre os provedores.

O que tudo isso nos diz é que calcular uma disponibilidade teórica máxima provavelmente produzirá apenas um cálculo aproximado da ordem de magnitude, mas, por si só, provavelmente não será preciso ou fornecerá uma visão significativa. O que a matemática nos diz é que quanto menos coisas dependerem de sua workload, reduzirá a probabilidade geral de falha. Quanto menos números menores que um forem multiplicados, maior será o resultado.

> Rule3
>
> Reduzir as dependências pode ter um impacto positivo na disponibilidade.

A matemática também ajuda a informar o processo de seleção de dependências. O processo de seleção afeta a forma como você projeta sua workload, como você aproveita a redundância nas dependências para melhorar sua disponibilidade e se você considera essas dependências como flexíveis ou rígidas. As dependências que podem afetar sua workload devem ser escolhidas com cuidado. A próxima regra fornece orientação sobre como fazer isso.

> Regra 4
>
> Em geral, selecione dependências cujas metas de disponibilidade sejam iguais ou maiores que as metas da sua workload.

Disponibilidade com redundância

Quando uma workload utiliza subsistemas múltiplos, independentes e redundantes, ela pode atingir um nível mais alto de disponibilidade teórica do que usando um único subsistema. Por exemplo, considere uma workload composta por dois subsistemas idênticos. Ele pode estar completamente operacional se o subsistema um ou o subsistema dois estiverem operacionais. Para que todo o sistema fique inativo, os dois subsistemas devem estar inativos ao mesmo tempo.

Se a probabilidade de falha de um subsistema for $1 - \alpha$, então a probabilidade de dois subsistemas redundantes estarem inativos ao mesmo tempo é o produto da probabilidade de falha de cada subsistema, $F = (1-\alpha_1) \times (1-\alpha_2)$. Para uma workload com dois subsistemas redundantes, usando a Equação (3), isso fornece uma disponibilidade definida como:

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

Equação 5

Portanto, para dois subsistemas cuja disponibilidade é de 99%, a probabilidade de um falhar é de 1% e a probabilidade de ambos falharem é (1−99%) × (1−99%) = 0,01%. Isso torna a disponibilidade usando dois subsistemas redundantes de 99,99%.

Isso também pode ser generalizado para incorporar peças de reposição redundantes adicionais s. Na Equação (5), assumimos apenas uma única peça de reposição, mas uma workload pode ter duas, três ou mais peças de reposição para que possa sobreviver à perda simultânea de vários subsistemas sem afetar a disponibilidade. Se uma workload tem três subsistemas e dois são de reposição, a probabilidade de que todos os três subsistemas falhem ao mesmo tempo é (1−α) × (1−α) × (1−α) or $(1-\alpha)^3$. Em geral, uma workload com s peças de reposição só falhará se os subsistemas s + 1 falharem.

Para uma workload com n subsistemas e s peças de reposição, f é o número de modos de falha ou as maneiras pelas quais os subsistemas s +1 podem falhar a partir de n.

Isso é efetivamente o teorema binomial, a matemática combinatória de escolher k elementos de um conjunto de n, ou “n escolher k”. Nesse caso, k é s + 1.

$$k = s + 1$$

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

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

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

Equação 6

Podemos então produzir uma aproximação de disponibilidade generalizada que incorpora o número de modos de falha e a economia. (Para entender por que isso é uma aproximação, consulte o Apêndice 2 de Highleyman, et al. Quebrando a barreira da disponibilidade.)

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

Equação 7

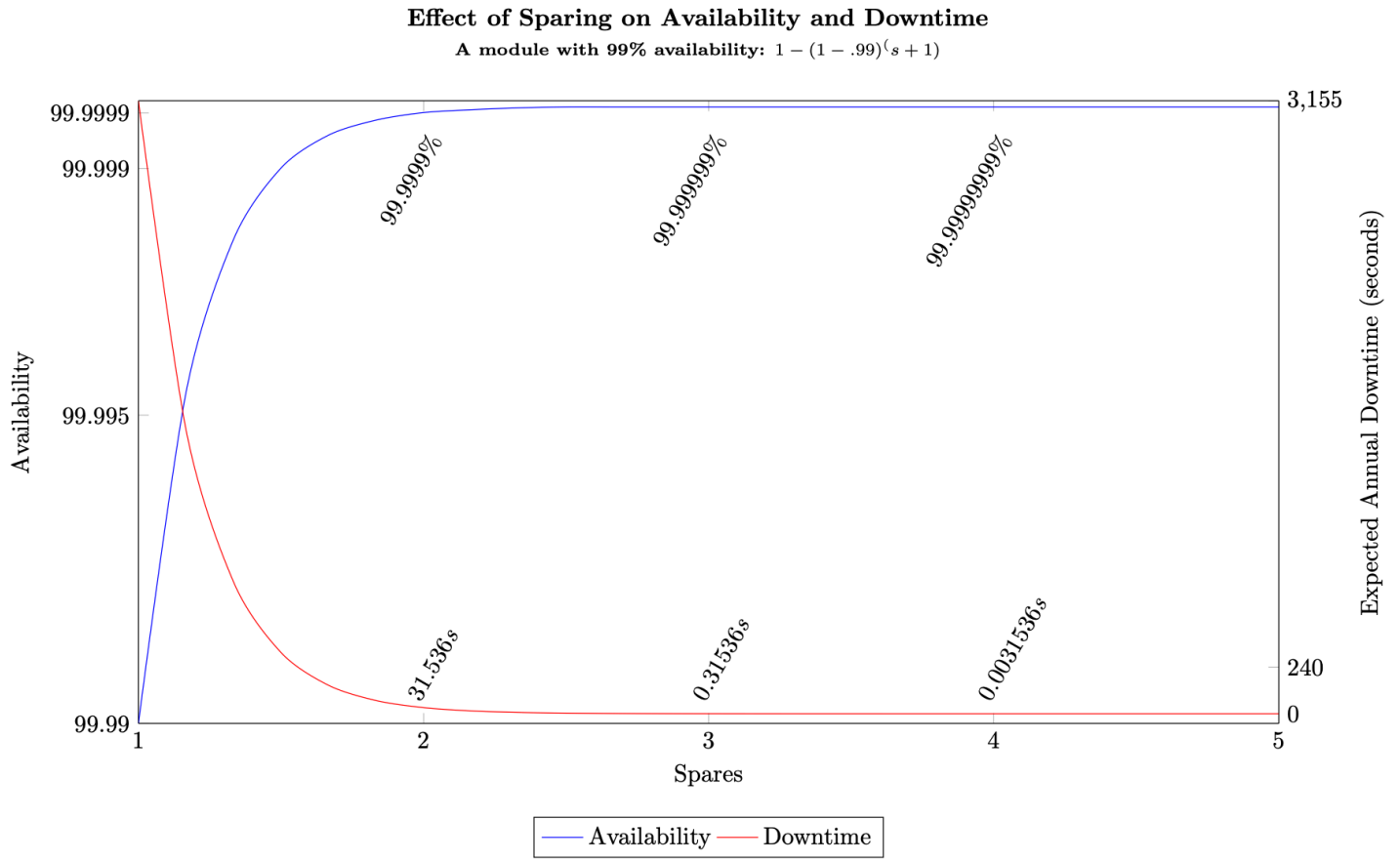
A economia pode ser aplicada a qualquer dependência que forneça recursos que falhem de forma independente. Instâncias do Amazon EC2 em diferentes AZs ou buckets do Amazon S3 em Regiões da AWS diferentes são exemplos disso. O uso de peças de reposição ajuda essa dependência a alcançar uma maior disponibilidade total para suportar as metas de disponibilidade da workload.

> **Regra 5**
>
> Use a economia para aumentar a disponibilidade das dependências em uma workload.

No entanto, economizar tem um custo. Cada reposição adicional custa o mesmo que o módulo original, aumentando o custo pelo menos linearmente. Criar uma workload que possa usar peças de reposição também aumenta sua complexidade. Ele deve saber como identificar falhas de dependência, transformar o trabalho em um recurso saudável e gerenciar a capacidade geral da workload.

A redundância é um problema de otimização. Poucas peças de reposição e a workload pode falhar com mais frequência do que o desejado, muitas peças sobressalentes e a workload custa muito para ser executada. Há um limite no qual adicionar mais peças de reposição custará mais do que a disponibilidade adicional que elas obtêm garante.

Usando nossa fórmula de disponibilidade geral com peças de reposição, Equação (7), para um subsistema que tem uma disponibilidade de 99,5%, com duas peças de reposição, a disponibilidade da workload é $A \approx 1 - (1)\ (1-0{,}95)^{3} = 99{,}9999875\%$ (aproximadamente 3,94 segundos de tempo de inatividade por ano), e com 10 peças de reposição, obtemos $A \approx 1 - (1)\ (1-0{,}995)^{11} = 25{,}5$ 9′s (o tempo de inatividade aproximado seria $1{,}26252 \times 10^{-15}$ms por ano, efetivamente 0). Ao comparar essas duas workloads, incorremos em um aumento de 5 vezes no custo de economia para obter quatro segundos a menos de tempo de inatividade por ano. Para a maioria das workloads, o aumento no custo seria injustificado devido a esse aumento na disponibilidade. A figura a seguir mostra essa relação.



Retornos decrescentes decorrentes do aumento da poupança

Com três peças de reposição ou mais, o resultado são frações de segundo do tempo de inatividade esperado por ano, o que significa que, após esse ponto, você atinge a área de retornos decrescentes. Pode haver uma necessidade de “adicionar mais” para alcançar níveis mais altos de disponibilidade, mas, na realidade, o custo-benefício desaparece muito rapidamente. O uso de mais de três peças de reposição não fornece um ganho material perceptível para quase todas as workloads quando o próprio subsistema tem pelo menos 99% de disponibilidade.

> **Regra 6**
>
> Há um limite superior para a eficiência de custos da poupança. Utilize o mínimo de peças de reposição necessárias para alcançar a disponibilidade necessária.

Você deve considerar a unidade de falha ao selecionar o número correto de peças de reposição. Por exemplo, vamos examinar uma workload que requer 10 instâncias do EC2 para lidar com a capacidade de pico e elas são implantadas em uma única AZ.

Como as AZs foram projetadas para serem limites de isolamento de falhas, a unidade de falha não é apenas uma única instância do EC2, porque uma AZ inteira de instâncias do EC2 pode falhar em conjunto. Nesse caso, você desejará adicionar redundância com outra AZ, implantando 10 instâncias EC2 adicionais para lidar com a carga em caso de falha de AZ, totalizando 20 instâncias EC2 (seguindo o padrão de estabilidade estática).

Embora pareçam ser 10 instâncias extras do EC2, na verdade é apenas uma única AZ sobressalente, portanto, não excedemos o ponto de retornos decrescentes. No entanto, você pode ser ainda mais econômico e, ao mesmo tempo, aumentar sua disponibilidade utilizando três AZs e implantando cinco instâncias EC2 por AZ.

Isso fornece uma AZ extra com um total de 15 instâncias EC2 (versus duas AZs com 20 instâncias), ainda fornecendo o total de 10 instâncias necessárias para atender à capacidade máxima durante um evento que afeta uma única AZ. Portanto, você deve incorporar o sparing para ser tolerante a falhas em todos os limites de isolamento de falhas usados pela workload (instância, célula, AZ e região).

Teorema CAP

Outra forma de pensarmos sobre a disponibilidade é em relação ao teorema CAP. O teorema afirma que um sistema distribuído, composto por vários nós que armazenam dados, não pode fornecer simultaneamente mais de duas das três garantias a seguir:

- Consistência: cada solicitação de leitura recebe a gravação mais recente ou um erro quando a consistência não pode ser garantida.
- Disponibilidade: cada solicitação recebe uma resposta sem erros, mesmo quando os nós estão inativos ou indisponíveis.
- Tolerância à partição: o sistema continua operando apesar da perda de um número arbitrário de mensagens entre os nós.

(Para obter mais detalhes, consulte Seth Gilbert e Nancy Lynch, A conjectura de Brewer e a viabilidade de serviços web consistentes, disponíveis e tolerantes à partição, ACM SIGACT News, Volume 33, Edição 2 (2002), pág. 51–59.)

A maioria dos sistemas distribuídos precisa tolerar falhas de rede e, portanto, o particionamento de rede deve ser permitido. Isso significa que essas workloads precisam escolher entre consistência e disponibilidade quando ocorre uma partição de rede. Se a workload escolher a disponibilidade,

ela sempre retornará uma resposta, mas com dados potencialmente inconsistentes. Se escolher a consistência, então, durante uma partição de rede, ele retornará um erro, pois a workload não pode ter certeza sobre a consistência dos dados.

Para workloads cujo objetivo é fornecer níveis mais altos de disponibilidade, elas podem escolher Disponibilidade e Tolerância à Partição (AP) para evitar o retorno de erros (estar indisponível) durante uma partição de rede. Isso resulta na necessidade de um modelo de consistência mais relaxado, como consistência eventual ou consistência monotônica.

Tolerância a falhas e isolamento de falhas

Esses são dois conceitos importantes quando pensamos em disponibilidade. A tolerância a falhas é a capacidade de resistir a falhas do subsistema e manter a disponibilidade (fazendo a coisa certa dentro de um SLA estabelecido). Para implementar a tolerância a falhas, as workloads usam subsistemas sobressalentes (ou redundantes). Quando um dos subsistemas em um conjunto redundante falha, outro retoma seu trabalho, normalmente quase sem problemas. Nesse caso, as peças de reposição são realmente capacidade ociosa; elas estão disponíveis para assumir 100% do trabalho do subsistema com falha. Com peças de reposição verdadeiras, várias falhas no subsistema são necessárias para produzir um impacto adverso na workload.

O isolamento de falhas minimiza o escopo do impacto quando ocorre uma falha. Isso normalmente é implementado com modularização. As workloads são divididas em pequenos subsistemas que falham de forma independente e podem ser reparados isoladamente. A falha de um módulo não se propaga além do módulo. Essa ideia se estende tanto verticalmente, em diferentes funcionalidades em uma workload, quanto horizontalmente, em vários subsistemas que fornecem a mesma funcionalidade. Esses módulos atuam como contêineres de falhas que limitam o escopo do impacto durante um evento.

Os padrões arquitetônicos de ambientes de gerenciamento, planos de dados e estabilidade estática apoiam diretamente a implementação de tolerância e isolamento de falhas. O artigo Estabilidade estática usando zonas de disponibilidade da Amazon Builders' Library fornece boas definições para esses termos e como eles se aplicam à criação de workloads resilientes e altamente disponíveis. Este whitepaper usa esses padrões na seção Projetando sistemas distribuídos altamente disponíveis no AWS, e também resumimos suas definições aqui.

- Ambiente de gerenciamento — A parte da workload envolvida na realização de alterações: adição de recursos, exclusão de recursos, modificação de recursos e propagação dessas alterações para onde elas são necessárias. Os ambientes de gerenciamento geralmente são mais complexos

e têm mais partes móveis do que os planos de dados e, portanto, são estatisticamente mais propensos a falhar e ter menores disponibilidades.

- Plano de dados — A parte da workload que fornece a funcionalidade comercial diária. Os planos de dados tendem a ser mais simples e operar em volumes maiores do que os ambientes de gerenciamento, levando a maiores disponibilidades.
- Estabilidade estática — A capacidade de uma workload continuar operando corretamente apesar das deficiências de dependência. Um método de implementação é remover as dependências do ambiente de gerenciamento dos planos de dados. Outro método é acoplar vagamente as dependências da workload. Talvez a workload não veja nenhuma informação atualizada (como coisas novas, coisas excluídas ou coisas modificadas) que sua dependência deveria ter fornecido. No entanto, tudo o que estava fazendo antes de a dependência ser prejudicada continua funcionando.

Quando pensamos no comprometimento de uma workload, há duas abordagens de alto nível que podemos considerar para a recuperação. O primeiro método é responder a essa deficiência depois que ela acontece, talvez usando o AWS Auto Scaling para adicionar nova capacidade. O segundo método é se preparar para essas deficiências antes que elas aconteçam, talvez superprovisionando a infraestrutura de uma workload para que ela possa continuar operando corretamente sem precisar de recursos adicionais.

Um sistema estaticamente estável usa a última abordagem. Ele pré-provisiona a capacidade não utilizada para estar disponível durante a falha. Esse método evita criar uma dependência em um ambiente de gerenciamento no caminho de recuperação da workload para provisionar nova capacidade de recuperação da falha. Além disso, o provisionamento de nova capacidade para vários recursos leva tempo. Enquanto espera por uma nova capacidade, sua workload pode ser sobrecarregada pela demanda existente e sofrer uma maior degradação, levando ao “esgotamento” ou à perda total da disponibilidade. No entanto, você também deve considerar as implicações de custo da utilização da capacidade pré-provisionada em relação às suas metas de disponibilidade.

A estabilidade estática fornece as próximas duas regras para workloads de alta disponibilidade.

> **Regra 7**
>
> Não use dependências em ambientes de gerenciamento em seu plano de dados, especialmente durante a recuperação.

Regra 8

Acople as dependências de forma flexível para que sua workload possa operar corretamente apesar do comprometimento da dependência, sempre que possível.

Medindo a disponibilidade

Como vimos anteriormente, criar um modelo de disponibilidade voltado para o futuro para um sistema distribuído é difícil e pode não fornecer os insights desejados. O que pode ser mais útil é desenvolver formas consistentes de medir a disponibilidade de sua workload.

A definição de disponibilidade como tempo de atividade e tempo de inatividade representa a falha como uma opção binária, ou a workload está ativa ou não.

No entanto, isso raramente é o caso. A falha tem um grau de impacto e geralmente ocorre em algum subconjunto da workload, afetando uma porcentagem de usuários ou solicitações, uma porcentagem de locais ou um percentual de latência. Todos esses são modos de falha parcial.

E embora o MTTR e o MTBF sejam úteis para entender o que impulsiona a disponibilidade resultante de um sistema e, portanto, como melhorá-lo, sua utilidade não é uma medida empírica de disponibilidade. Além disso, as workloads são compostas por vários componentes. Por exemplo, uma workload, como um sistema de processamento de pagamentos, é composta por várias interfaces de programação de aplicativos (APIs) e subsistemas. Então, quando queremos fazer uma pergunta como “qual é a disponibilidade de toda a workload?” , na verdade, é uma pergunta complexa e cheia de nuances.

Nesta seção, examinaremos três maneiras pelas quais a disponibilidade pode ser medida empiricamente: taxa de sucesso da solicitação do lado do servidor, taxa de sucesso da solicitação do lado do cliente e tempo de inatividade anual.

Taxa sobre o sucesso das solicitações nos lados do servidor e do cliente

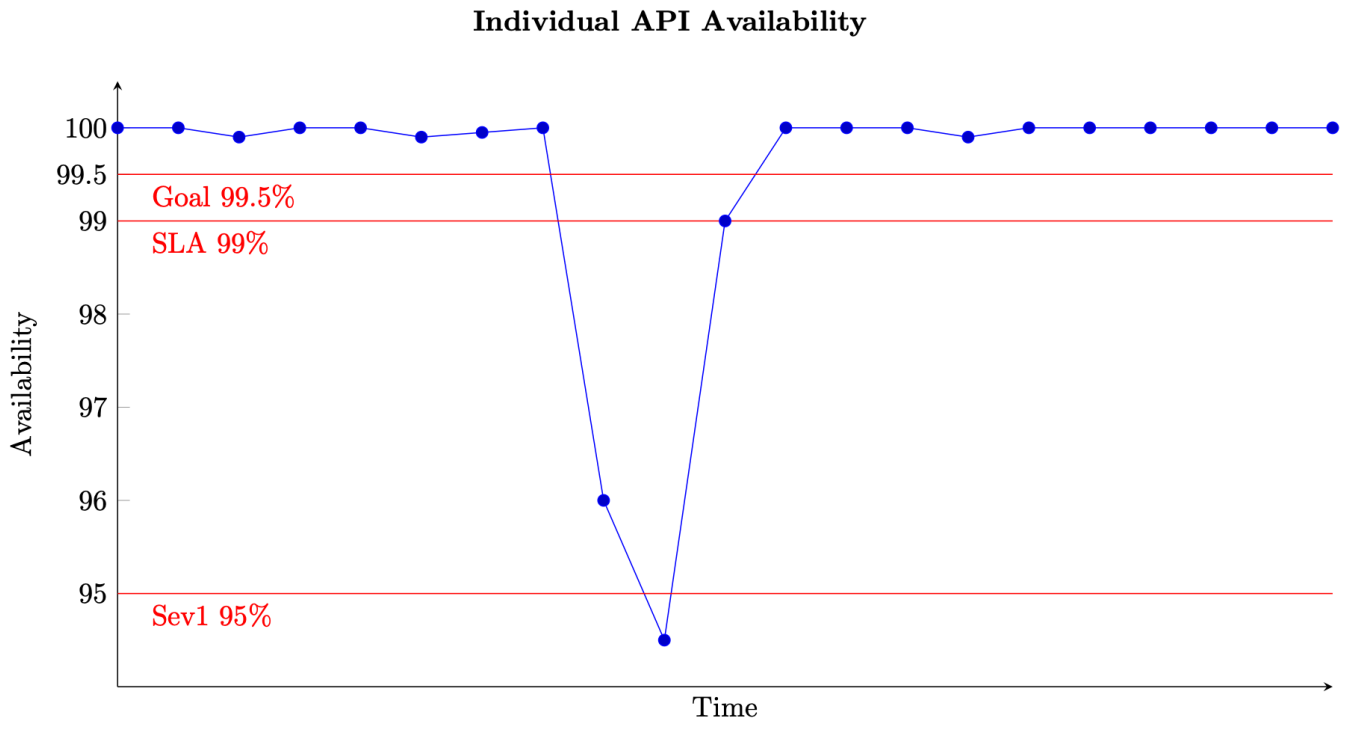
Os dois primeiros métodos são muito semelhantes, diferindo apenas do ponto de vista em que a medição é feita. As métricas do lado do servidor podem ser coletadas da instrumentação do serviço. No entanto, eles não estão completos. Se os clientes não conseguirem acessar o serviço, você não conseguirá coletar essas métricas. Para entender a experiência do cliente, em vez de confiar na telemetria dos clientes sobre solicitações malsucedidas, uma maneira mais fácil de coletar métricas do lado do cliente é simular o tráfego do cliente com os canários, um software que examina regularmente seus serviços e registra métricas.

Esses dois métodos calculam a disponibilidade como a fração do total de unidades de trabalho válidas que o serviço recebe e as que ele processa com sucesso (isso ignora unidades de trabalho inválidas, como uma solicitação HTTP que resulta em um erro 404).

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

Equação 8

Para um serviço baseado em solicitações, a unidade de trabalho é a solicitação, como uma solicitação HTTP. Para serviços baseados em eventos ou tarefas, as unidades de trabalho são eventos ou tarefas, como processar uma mensagem fora de uma fila. Essa medida de disponibilidade é significativa em intervalos curtos, como janelas de um ou cinco minutos. Também é mais adequado em uma perspectiva granular, como em um nível por API para um serviço baseado em solicitações. A figura a seguir fornece uma visão de como pode ser a disponibilidade ao longo do tempo quando calculada dessa forma. Cada ponto de dados no gráfico é calculado usando a Equação (8) em uma janela de cinco minutos (você pode escolher outras dimensões de tempo, como intervalos de um minuto ou dez minutos). Por exemplo, o ponto de dados 10 mostra 94,5% de disponibilidade. Isso significa que, durante os minutos t+45 a t+50, se o serviço recebeu 1.000 solicitações, somente 945 delas foram processadas com sucesso.



Exemplo de medição da disponibilidade ao longo do tempo para uma única API

O gráfico também mostra a meta de disponibilidade da API, disponibilidade de 99,5%, o acordo de serviço (SLA) que ela oferece aos clientes, 99% de disponibilidade e o limite para um alarme de alta severidade, 95%. Sem o contexto desses diferentes limites, um gráfico de disponibilidade pode não fornecer uma visão significativa de como seu serviço está operando.

Também queremos ser capazes de rastrear e descrever a disponibilidade de um subsistema maior, como um ambiente de gerenciamento ou um serviço inteiro. Uma maneira de fazer isso é obter a média de cada ponto de dados de cinco minutos para cada subsistema. O gráfico será semelhante ao anterior, mas representará um conjunto maior de entradas. Também dá peso igual a todos os subsistemas que compõem seu serviço. Uma abordagem alternativa pode ser somar todas as solicitações recebidas e processadas com sucesso de todas as APIs do serviço para calcular a disponibilidade em intervalos de cinco minutos.

No entanto, esse último método pode ocultar uma API individual com baixo throughput e baixa disponibilidade. Como exemplo simples, considere um serviço com duas APIs.

A primeira API recebe 1.000.000 de solicitações em uma janela de cinco minutos e processa com sucesso 999.000 delas, oferecendo uma disponibilidade de 99,9%. A primeira API recebe 100 de solicitações na mesma janela de cinco minutos e processa com sucesso apenas 50 delas, oferecendo uma disponibilidade de 50%.

Se somarmos as solicitações de cada API, haverá 1.000.100 solicitações válidas no total e 999.050 delas serão processadas com sucesso, oferecendo uma disponibilidade geral de 99,895% para o serviço. Mas, se calcularmos a média das disponibilidades das duas APIs, o método anterior, obteremos uma disponibilidade resultante de 74,95%, o que pode ser mais revelador da experiência real.

Nenhuma abordagem está errada, mas mostra a importância de entender o que as métricas de disponibilidade estão lhe dizendo. Você pode optar por preferir a soma de solicitações para todos os subsistemas se sua workload receber um volume de solicitações semelhante em cada um. Essa abordagem se concentra na “solicitação” e em seu sucesso como medida da disponibilidade e da experiência do cliente. Como alternativa, você pode optar por calcular a média das disponibilidades do subsistema para representar igualmente sua criticidade, apesar das diferenças no volume de solicitações. Essa abordagem se concentra no subsistema e na capacidade de cada um como um substituto para a experiência do cliente.

Tempo de inatividade anual

A terceira abordagem é calcular o tempo de inatividade anual. Essa forma de métrica de disponibilidade é mais apropriada para definição e revisão de metas de longo prazo. Isso exige definir o que significa tempo de inatividade para sua workload. Em seguida, você pode medir a disponibilidade com base no número de minutos em que a workload não esteve em uma condição de "interrupção" em relação ao número total de minutos no período determinado.

Algumas workloads podem definir o tempo de inatividade como algo como uma queda abaixo de 95% da disponibilidade de uma única API ou função de workload por um intervalo de um ou cinco minutos (o que ocorreu no gráfico de disponibilidade anterior). Você também pode considerar apenas o tempo de inatividade, pois ele se aplica a um subconjunto de operações críticas do plano de dados. Por exemplo, o Acordo de Nível de Serviço do Amazon Messaging (SQS, SNS) para disponibilidade do SQS se aplica à API de envio, recebimento e exclusão do SQS.

Workloads maiores e mais complexas talvez precisem definir métricas de disponibilidade em todo o sistema. Para um grande site de comércio eletrônico, uma métrica de todo o sistema pode ser algo como a taxa de pedidos do cliente. Aqui, uma queda de 10% ou mais nos pedidos em comparação com a quantidade prevista durante qualquer janela de cinco minutos pode definir o tempo de inatividade.

Em qualquer uma das abordagens, você pode então somar todos os períodos de interrupção para calcular uma disponibilidade anual. Por exemplo, se durante um ano civil houve 27 períodos de inatividade de cinco minutos, definidos como a disponibilidade de qualquer API do plano de dados caindo abaixo de 95%, o tempo de inatividade geral foi de 135 minutos (alguns períodos de cinco minutos podem ter sido consecutivos, outros isolados), representando uma disponibilidade anual de 99,97%.

Esse método adicional de medir a disponibilidade pode fornecer dados e insights ausentes nas métricas do lado do cliente e do lado do servidor. Por exemplo, considere uma workload prejudicada e com taxas de erro significativamente elevadas. Os clientes dessa workload podem parar completamente de fazer chamadas para seus serviços. Talvez eles tenham ativado um disjuntor ou seguido seu plano de recuperação de desastres para usar o serviço em uma região diferente. Se estivéssemos medindo apenas as respostas falhadas, a disponibilidade da workload pode realmente aumentar durante a deficiência, mas não porque a deficiência melhora ou desaparece, mas porque os clientes simplesmente param de usá-la.

Latência

Por fim, também é importante medir a latência das unidades de trabalho de processamento em sua workload. Parte da definição de disponibilidade é fazer o trabalho dentro de um SLA estabelecido. Se o retorno de uma resposta demorar mais do que o tempo limite do cliente, a percepção do cliente é de que a solicitação falhou e a workload não está disponível. No entanto, no lado do servidor, a solicitação pode parecer ter sido processada com sucesso.

A medição da latência fornece outra lente para avaliar a disponibilidade. Usar percentis e média reduzida são boas estatísticas para essa medição. Eles são comumente medidos no percentil 50 (P50 e TM50) e no percentil 99 (P99 e TM99). A latência deve ser medida com canários para representar a experiência do cliente, bem como com métricas do lado do servidor. Sempre que a média de algum percentual de latência, como P99 ou TM99.9, ultrapassar o SLA desejado, considere esse tempo de inatividade, que contribui para o cálculo anual do tempo de inatividade.

Projetando sistemas distribuídos de alta disponibilidade no AWS

As seções anteriores trataram principalmente da disponibilidade teórica das workloads e do que elas podem alcançar. Eles são um conjunto importante de conceitos que você deve ter em mente ao criar sistemas distribuídos. Eles ajudarão a informar seu processo de seleção de dependências e como você implementa a redundância.

Também analisamos a relação do MTTD, MTTR e MTBF com a disponibilidade. Esta seção apresentará orientações práticas com base na teoria anterior. Em resumo, as workloads de engenharia para alta disponibilidade visam aumentar o MTBF e diminuir o MTTR e o MTTD.

Embora eliminar todas as falhas seja o ideal, isso não é realista. Em grandes sistemas distribuídos com dependências profundamente empilhadas, ocorrerão falhas. “Tudo falha o tempo todo” (veja Werner Vogels, CTO da Amazon.com, 10 lições de 10 anos de Amazon Web Services.) e “você não pode legislar contra falhas [então] foque na detecção e resposta rápidas”. (consulte Chris Pinkham, membro fundador da equipe do Amazon EC2, Projeto de ARC335 para falhas: arquitetando sistemas resilientes no AWS)

O que isso significa é que, frequentemente, você não tem controle sobre se a falha acontece. O que você pode controlar é a rapidez com que detecta a falha e faz algo a respeito. Portanto, embora o aumento do MTBF ainda seja um componente importante da alta disponibilidade, a mudança mais significativa que os clientes têm sob seu controle é a redução do MTTD e do MTTR.

Tópicos

- Redução do MTTD
- Redução do MTTD
- Aumento do MTBF

Redução do MTTD

Reduzir o MTTD de uma falha significa descobrir a falha o mais rápido possível. A redução do MTTD é baseada na observabilidade ou na forma como você instrumentou sua workload para entender seu estado. Os clientes devem monitorar suas métricas de experiência do cliente nos subsistemas críticos da workload como forma de identificar proativamente quando ocorre um problema (consulte o Apêndice 1 — Métricas críticas de MTTD e MTTR para obter mais informações sobre essas

métricas).). Os clientes podem usar o Amazon CloudWatch Synthetics para criar canários que monitoram suas APIs e consoles para medir proativamente a experiência do usuário. Há vários outros mecanismos de verificação de integridade que podem ser usados para minimizar o MTTD, como verificações de integridade do Elastic Load Balancing (ELB), verificações de integridade do Amazon Route 53 e muito mais. (Consulte Amazon Builders' Library — Implementando verificações de integridade.)

Seu monitoramento também precisa ser capaz de detectar falhas parciais do sistema como um todo e em seus subsistemas individuais. Suas métricas de disponibilidade, falha e latência devem usar a dimensionalidade dos limites de isolamento de falhas como dimensões métricas do CloudWatch. Por exemplo, considere uma única instância do EC2 que faz parte de uma arquitetura baseada em células, na AZ use1-az1, na região us-east-1, que faz parte da API de atualização da workload que faz parte de seu subsistema de ambiente de gerenciamento. Quando o servidor envia suas métricas, ele pode usar o ID da instância, AZ, região, nome da API e nome do subsistema como dimensões. Isso permite que você tenha observabilidade e defina alarmes em cada uma dessas dimensões para detectar falhas.

Redução do MTTD

Depois que uma falha é descoberta, o restante do tempo de MTTR é o reparo real ou a mitigação do impacto. Para reparar ou mitigar uma falha, você precisa saber o que está errado. Há dois grupos principais de métricas que fornecem informações durante essa fase: métricas de Avaliação de Impacto e métricas de Saúde Operacional. O primeiro grupo informa o escopo do impacto durante uma falha, medindo o número ou a porcentagem dos clientes, recursos ou workloads afetados. O segundo grupo ajuda a identificar por que há impacto. Depois que o motivo é descoberto, os operadores e a automação podem responder e resolver a falha. Consulte o Apêndice 1 — Métricas críticas de MTTD e MTTR para obter mais informações sobre essas métricas.

> **Regra 9**
>
> A observabilidade e a instrumentação são fundamentais para reduzir o MTTD e o MTTR.

Rota para contornar falhas

A abordagem mais rápida para mitigar o impacto é por meio de subsistemas que evitam falhas. Essa abordagem usa redundância para reduzir o MTTR transferindo rapidamente o trabalho de um

subsistema com falha para um sobressalente. A redundância pode variar de processos de software a instâncias do EC2, a várias AZs e a várias regiões.

Subsistemas de reposição podem reduzir o MTTR a quase zero. O tempo de recuperação é apenas o necessário para redirecionar o trabalho para o sobressalente em espera. Isso geralmente acontece com latência mínima e permite que o trabalho seja concluído dentro do SLA definido, mantendo a disponibilidade do sistema. Isso produz MTTRs que são experimentados como atrasos leves, talvez até imperceptíveis, em vez de períodos prolongados de indisponibilidade.

Por exemplo, se seu serviço utiliza instâncias do EC2 por trás de um Application Load Balancer (ALB), você pode configurar verificações de integridade em um intervalo de até cinco segundos e exigir apenas duas verificações de integridade com falha antes que um destino seja marcado como não íntegro. Isso significa que, em 10 segundos, você pode detectar uma falha e parar de enviar tráfego para o host não íntegro. Nesse caso, o MTTR é efetivamente o mesmo que o MTTD, pois assim que a falha é detectada, ela também é mitigada.

É isso que as workloads de alta disponibilidade ou disponibilidade contínua estão tentando alcançar. Queremos contornar rapidamente as falhas na workload detectando rapidamente os subsistemas com falha, marcando-os como falhados, parando de enviar tráfego para eles e, em vez disso, enviá-lo para um subsistema redundante.

Observe que o uso desse tipo de mecanismo rápido torna sua workload muito sensível a erros transitórios. No exemplo fornecido, certifique-se de que as verificações de integridade do balanceador de carga estejam realizando verificações de integridade superficiais ou dinâmicas e locais apenas da instância, sem testar dependências ou fluxos de trabalho (geralmente chamados de verificações de integridade profundas). Isso ajudará a evitar a substituição desnecessária de instâncias durante erros transitórios que afetam a workload.

A observabilidade e a capacidade de detectar falhas em subsistemas são essenciais para que o roteamento em torno da falha seja bem-sucedido. Você precisa conhecer o escopo do impacto para que os recursos afetados possam ser marcados como insalubres ou com falha e retirados de serviço para que possam ser encaminhados. Por exemplo, se uma única AZ apresentar uma deficiência parcial no serviço, sua instrumentação precisará ser capaz de identificar que há um problema localizado na AZ para contornar todos os recursos dessa AZ até que ela se recupere.

Ser capaz de contornar falhas também pode exigir ferramentas adicionais, dependendo do ambiente. Usando o exemplo anterior com instâncias do EC2 por trás de um ALB, imagine que instâncias em uma AZ possam estar passando por verificações de integridade locais, mas uma deficiência isolada de AZ esteja fazendo com que elas não consigam se conectar ao banco de dados em

outra AZ. Nesse caso, as verificações de integridade do balanceamento de carga não tirarão essas instâncias de serviço. Seria necessário um mecanismo automatizado diferente para remover a AZ do balanceador de carga ou forçar as instâncias a falharem nas verificações de integridade, o que depende da identificação de que o escopo do impacto é uma AZ. Para workloads que não estão usando um balanceador de carga, seria necessário um método semelhante para impedir que recursos em uma AZ específica aceitassem unidades de trabalho ou removessem completamente a capacidade da AZ.

Em alguns casos, a mudança de trabalho para um subsistema redundante não pode ser automatizada, como o failover de um banco de dados primário para o secundário, em que a tecnologia não fornece sua própria eleição de líder. Esse é um cenário comum para arquiteturas multirregionais do AWS. Como esses tipos de failovers exigem algum tempo de inatividade para serem realizados, não podem ser revertidos imediatamente e deixam a workload sem redundância por um período de tempo, é importante ter uma pessoa no processo de tomada de decisão.

Workloads que podem adotar um modelo de consistência menos rigoroso podem obter MTTRs mais curtos usando a automação de failover multirregional para contornar falhas. Recursos como a replicação entre regiões do Amazon S3 ou as tabelas globais do Amazon DynamoDB fornecem recursos multirregionais por meio de replicação eventualmente consistente. Além disso, usar um modelo de consistência relaxado é benéfico quando consideramos o teorema CAP. Durante falhas de rede que afetam a conectividade com subsistemas com estado, se a workload escolher a disponibilidade em vez da consistência, ela ainda poderá fornecer respostas sem erros, outra forma de contornar a falha.

O roteamento em caso de falha pode ser implementado com duas estratégias diferentes. A primeira estratégia é implementar a estabilidade estática pré-provisionando recursos suficientes para lidar com a carga completa do subsistema com falha. Isso pode ser uma única instância do EC2 ou pode ser uma AZ inteira em capacidade. A tentativa de provisionar novos recursos durante uma falha aumenta o MTTR e adiciona uma dependência a um ambiente de gerenciamento em seu caminho de recuperação. No entanto, isso tem um custo adicional.

A segunda estratégia é rotear parte do tráfego do subsistema com falha para outros e eliminar a carga do excesso de tráfego que não pode ser tratado pela capacidade restante. Durante esse período de degradação, você pode ampliar novos recursos para substituir a capacidade com falha. Essa abordagem tem um MTTR mais longo e cria uma dependência em um ambiente de gerenciamento, mas custa menos em capacidade ociosa em espera.

Retorne a um bom estado conhecido

Outra abordagem comum para mitigação durante o reparo é retornar a workload a um estado anterior conhecido como bom. Se uma alteração recente pode ter causado a falha, reverter essa alteração é uma forma de retornar ao estado anterior.

Em outros casos, condições transitórias podem ter causado a falha; nesse caso, reiniciar a workload pode mitigar o impacto. Vamos examinar esses dois cenários.

Durante uma implantação, minimizar o MTTD e o MTTR depende da observabilidade e da automação. Seu processo de implantação deve observar continuamente a workload para detectar a introdução de maiores taxas de erro, maior latência ou anomalias. Depois que eles forem reconhecidos, o processo de implantação deverá ser interrompido.

Existem várias estratégias de implantação, como implantações no local, implantações azul/verdes e implantações contínuas. Cada um deles pode usar um mecanismo diferente para retornar a um estado em boas condições. Ele pode voltar automaticamente para o estado anterior, mudar o tráfego de volta para o ambiente azul ou exigir intervenção manual.

O CloudFormation oferece a capacidade de reverter automaticamente como parte de suas operações de criação e atualização de pilha, assim como o AWS CodeDeploy. O CodeDeploy também suporta implantações azul/verde e contínuas.

Para aproveitar esses recursos e minimizar seu MTTR, considere automatizar todas as suas implantações de infraestrutura e código por meio desses serviços. Em cenários em que você não pode usar esses serviços, considere implementar o padrão de saga com o AWS Step Functions para reverter implantações com falha.

Ao considerar a reinicialização, existem várias abordagens diferentes. Elas variam desde a reinicialização de um servidor, a tarefa mais longa, até a reinicialização de um thread, a tarefa mais curta. Aqui está uma tabela que descreve algumas das abordagens de reinicialização e os tempos aproximados de conclusão (representativos da diferença de ordens de magnitude, eles não são exatos).

Mecanismo de recuperação de falhas	MTTR estimado
Iniciar e configurar o novo servidor virtual	15 minutos
Reimplantar o software	10 minutos

Mecanismo de recuperação de falhas	MTTR estimado
Reinicializar servidor	5 minutos
Reiniciar ou iniciar o contêiner	2 segundos
Invocar nova função sem servidor	100 ms
Reiniciar processo	10 ms
Reiniciar thread	10 µs

Analisando a tabela, há alguns benefícios claros do MTTR no uso de contêineres e funções sem servidor (como AWS Lambda). Seu MTTR é muito mais rápido do que reiniciar uma máquina virtual ou lançar uma nova. No entanto, usar o isolamento de falhas por meio da modularidade do software também é benéfico. Se você puder conter falhas em um único processo ou thread, a recuperação dessa falha é muito mais rápida do que reiniciar um contêiner ou servidor.

Como abordagem geral para a recuperação, você pode passar de baixo para cima: 1/Reiniciar, 2/Reiniciar, 3/Reimagem/Reimplantar, 4/Substituir. No entanto, quando você chega à etapa de reinicialização, contornar a falha geralmente é uma abordagem mais rápida (geralmente levando no máximo de 3 a 4 minutos). Portanto, para mitigar o impacto mais rapidamente após uma tentativa de reinicialização, contorne a falha e, em segundo plano, continue o processo de recuperação para devolver a capacidade à sua workload.

> Regra 10
>
> Concentre-se na mitigação do impacto, não na resolução de problemas. Escolha o caminho mais rápido de volta à operação normal.

Diagnóstico de falhas

Parte do processo de reparo após a detecção é o período de diagnóstico. Esse é o período em que os operadores tentam determinar o que está errado. Esse processo pode envolver a consulta de registros, a revisão de métricas de saúde operacional ou o login em hosts para solucionar problemas. Todas essas ações exigem tempo, portanto, criar ferramentas e runbooks para agilizar essas ações também pode ajudar a reduzir o MTTR.

Runbooks e automação

Da mesma forma, depois de determinar o que está errado e qual curso de ação reparará a workload, os operadores normalmente precisam executar um conjunto de etapas para fazer isso. Por exemplo, após uma falha, a maneira mais rápida de reparar a workload pode ser reiniciá-la, o que pode envolver várias etapas ordenadas. A utilização de um runbook que automatize essas etapas ou forneça orientação específica a um operador agilizará o processo e ajudará a reduzir o risco de ações inadvertidas.

Aumento do MTBF

O componente final para melhorar a disponibilidade é aumentar o MTBF. Isso pode se aplicar tanto ao software quanto aos serviços do AWS usados para executá-lo.

Aumentando o MTBF do sistema distribuído

Uma forma de aumentar o MTBF é reduzir os defeitos no software. Há várias maneiras de fazer isso. Os clientes podem usar ferramentas como o Amazon CodeGuru Reviewer para encontrar e corrigir erros comuns. Você também deve realizar análises abrangentes de código por pares, testes unitários, testes de integração, testes de regressão e testes de carga no software antes que ele seja implantado na produção. Aumentar a quantidade de cobertura de código nos testes ajudará a garantir que até mesmo caminhos incomuns de execução de código sejam testados.

A implantação de mudanças menores também pode ajudar a evitar resultados inesperados, reduzindo a complexidade da mudança. Cada atividade oferece a oportunidade de identificar e corrigir defeitos antes que eles possam ser invocados.

Outra abordagem para evitar falhas são os testes regulares. A implementação de um programa de engenharia de caos pode ajudar a testar como sua workload falha, validar procedimentos de recuperação e ajudar a encontrar e corrigir os modos de falha antes que eles ocorram na produção. Os clientes podem usar o AWS Fault Injection Simulator como parte de seu conjunto de ferramentas para experimentos de engenharia do caos.

A tolerância a falhas é outra forma de evitar falhas em um sistema distribuído. Módulos rápidos, novas tentativas com recuo e instabilidade exponenciais, transações e idempotência são técnicas para ajudar a tornar as workloads tolerantes a falhas.

As transações são um grupo de operações que aderem às propriedades ACID. Eles são os seguintes:

- Atomicidade — Ou todas as ações acontecem ou nenhuma delas acontecerá.
- Consistência — Cada transação deixa a workload em um estado válido.
- Isolamento — As transações realizadas simultaneamente deixam a workload no mesmo estado como se tivessem sido executadas sequencialmente.
- Durabilidade — Depois que uma transação é confirmada, todos os seus efeitos são preservados mesmo no caso de falha na workload.

Novas tentativas com recuo exponencial e instabilidade permitem que você supere falhas transitórias causadas por Heisenbugs, sobrecarga ou outras condições. Quando as transações são idempotentes, elas podem ser repetidas várias vezes sem efeitos colaterais.

Se considerarmos o efeito de um Heisenbug em uma configuração de hardware tolerante a falhas, não nos preocuparíamos, pois a probabilidade de o Heisenbug aparecer tanto no subsistema primário quanto no redundante é infinitesimalmente pequena. (Veja Jim Gray, "Por que os computadores param e o que pode ser feito a respeito? ", junho de 1985, Relatório Técnico Tandem 85.7.) Em sistemas distribuídos, queremos alcançar os mesmos resultados com nosso software.

Quando um Heisenbug é invocado, é fundamental que o software detecte rapidamente a operação incorreta e falhe para que possa ser tentado novamente. Isso é obtido por meio de programação defensiva e validação de entradas, resultados intermediários e saídas. Além disso, os processos são isolados e não compartilham nenhum estado com outros processos.

Essa abordagem modular garante que o escopo do impacto durante a falha seja limitado. Os processos falham de forma independente. Quando um processo falha, o software deve usar “pares de processos” para repetir o trabalho, o que significa que um novo processo pode assumir o trabalho de um que falhou. Para manter a confiabilidade e a integridade da workload, cada operação deve ser tratada como uma transação ACID.

Isso permite que um processo falhe sem corromper o estado da workload, abortando a transação e revertendo as alterações feitas. Isso permite que o processo de recuperação repita a transação a partir de um estado em boas condições e reinicie normalmente. É assim que o software pode ser tolerante a falhas para a Heisenbugs.

No entanto, você não deve tentar tornar o software tolerante a falhas para Bohrbugs. Esses defeitos devem ser encontrados e removidos antes que a workload entre em produção, pois nenhum nível de redundância jamais alcançará o resultado correto. (Veja Jim Gray, "Por que os computadores param e o que pode ser feito a respeito? ", junho de 1985, Relatório Técnico Tandem 85.7.)

A maneira final de aumentar o MTBF é reduzir o escopo do impacto da falha. Usar o isolamento de falhas por meio da modularização para criar contêineres de falhas é a principal maneira de fazer isso, conforme descrito anteriormente em Tolerância e isolamento de falhas. Reduzir a taxa de falhas melhora a disponibilidade. O AWS usa técnicas como divisão de serviços em ambientes de gerenciamento e planos de dados, independência da zona de disponibilidade (AZI), isolamento regional, arquiteturas baseadas em células e fragmentação aleatória para fornecer isolamento de falhas. Esses também são padrões que também podem ser usados pelos clientes do AWS.

Por exemplo, vamos analisar um cenário em que uma workload colocava os clientes em diferentes contêineres de falhas de sua infraestrutura, atendendo no máximo 5% do total de clientes. Um desses contêineres de falhas passa por um evento que aumenta a latência além do tempo limite do cliente em 10% das solicitações. Durante esse evento, para 95% dos clientes, o serviço estava 100% disponível. Para os outros 5%, o serviço parecia estar 90% disponível. Isso resulta em uma disponibilidade de 1 − (5% dos clientes × 10% de suas solicitações) = 99,5% em vez de 10% das solicitações falharem para 100% dos clientes (resultando em uma disponibilidade de 90%).

> Regra 11
>
> O isolamento de falhas diminui o escopo do impacto e aumenta o MTBF da workload ao reduzir a taxa geral de falhas.

Aumento da dependência MTBF

O primeiro método para aumentar seu MTBF de dependência do AWS é usar o isolamento de falhas. Muitos serviços do AWS oferecem um nível de isolamento na AZ, o que significa que uma falha em uma AZ não afeta o serviço em outra AZ.

O uso de instâncias EC2 redundantes em várias AZs aumenta a disponibilidade do subsistema. O AZI fornece uma capacidade de economia dentro de uma única região, permitindo que você aumente sua disponibilidade para os serviços do AZI.

No entanto, nem todos os serviços do AWS operam no nível AZ. Muitos outros oferecem isolamento regional. Nesse caso, em que a disponibilidade projetada do serviço regional não oferece suporte à disponibilidade geral necessária para sua workload, você pode considerar uma abordagem multirregional. Cada região oferece uma instanciação isolada do serviço, equivalente à economia.

Existem vários serviços que ajudam a facilitar a criação de um serviço multirregional. Por exemplo:

- Amazon Aurora Global Database
- Tabelas globais do Amazon DynamoDB
- Amazon ElastiCache para Redis:– datastore global
- AWS Global Accelerator
- Replicação entre regiões do Amazon S3
- Amazon Route 53 Application Recovery Controller

Este documento não aborda as estratégias de criação de workloads multirregionais, mas você deve avaliar os benefícios de disponibilidade das arquiteturas multirregionais com o custo adicional, a complexidade e as práticas operacionais necessárias para atingir as metas de disponibilidade desejadas.

O próximo método para aumentar o MTBF de dependência é projetar sua workload para ser estaticamente estável. Por exemplo, você tem uma workload que fornece informações do produto. Quando seus clientes fazem uma solicitação de um produto, seu serviço faz uma solicitação a um serviço externo de metadados para recuperar os detalhes do produto. Em seguida, sua workload retorna todas essas informações para o usuário.

No entanto, se o serviço de metadados não estiver disponível, as solicitações feitas por seus clientes falharão. Em vez disso, você pode extrair ou enviar de forma assíncrona os metadados localmente para o seu serviço para serem usados para responder às solicitações. Isso elimina a chamada síncrona para o serviço de metadados do seu caminho crítico.

Além disso, como seu serviço ainda está disponível mesmo quando o serviço de metadados não está, você pode removê-lo como uma dependência no cálculo de disponibilidade. Esse exemplo depende da suposição de que os metadados não mudam com frequência e que veicular metadados obsoletos é melhor do que a falha na solicitação. Outro exemplo semelhante é o serve-stale para DNS, que permite que os dados sejam mantidos no cache além da expiração do TTL e usados para respostas quando uma resposta atualizada não está prontamente disponível.

O método final de aumentar o MTBF da dependência é reduzir o escopo do impacto da falha. Conforme discutido anteriormente, a falha não é um evento binário, há graus de falha. Esse é o efeito da modularização; a falha está contida apenas nas solicitações ou usuários atendidos por esse contêiner.

Isso resulta em menos falhas durante um evento, o que, em última análise, aumenta a disponibilidade da workload geral ao limitar o escopo do impacto.

Reduzindo fontes comuns de impacto

Em 1985, Jim Gray descobriu, durante um estudo na Tandem Computers, que a falha era causada principalmente por duas coisas: software e operações. (Veja Jim Gray, "Por que os computadores param e o que pode ser feito a respeito? ", junho de 1985, Relatório Técnico Tandem 85.7.) Mesmo depois de 36 anos, isso continua sendo verdade. Apesar dos avanços na tecnologia, não há uma solução fácil para esses problemas, e as principais fontes de falha não mudaram. O tratamento de falhas no software foi discutido no início desta seção, portanto, o foco aqui será as operações e a redução da frequência de falhas.

Estabilidade em comparação com os recursos

Se nos referirmos às taxas de falha do gráfico de software e hardware na seçãothe section called "Disponibilidade do sistema distribuído", podemos observar que defeitos são adicionados em cada versão do software. Isso significa que qualquer alteração na workload aumenta o risco de falha. Essas mudanças geralmente são coisas como novos recursos, o que fornece um corolário. Workloads com maior disponibilidade favorecerão a estabilidade em relação aos novos recursos. Assim, uma das maneiras mais simples de melhorar a disponibilidade é implantar com menos frequência ou fornecer menos recursos. As workloads implantadas com mais frequência terão inerentemente uma disponibilidade menor do que aquelas que não o fazem. No entanto, as workloads que não adicionam recursos não atendem à demanda do cliente e podem se tornar menos úteis com o tempo.

Então, como continuamos inovando e lançando recursos com segurança? A resposta é padronização. Qual é a maneira correta de implantar? Como você solicita implantações? Quais são os padrões para testes? Quanto tempo você espera entre os estágios? Seus testes de unidade abrangem o suficiente do código do software? Essas são perguntas que a padronização responderá e evitará problemas causados por coisas como não testar a carga, pular estágios de implantação ou implantar muito rapidamente em muitos hosts.

A forma como você implementa a padronização é por meio da automação. Ele reduz a chance de erros humanos e permite que os computadores façam aquilo em que são bons, ou seja, fazer sempre a mesma coisa da mesma maneira. A maneira de unir padronização e automação é definir metas. Objetivos como não fazer alterações manuais, hospedar somente por meio de sistemas de autorização contingente, escrever testes de carga para cada API e assim por diante. A excelência operacional é uma norma cultural que pode exigir mudanças substanciais. Estabelecer e monitorar o desempenho em relação a uma meta ajuda a impulsionar uma mudança cultural que terá um

amplo impacto na disponibilidade da workload. O pilar AWS Well-Architected Operational Excellence fornece as melhores práticas abrangentes para a excelência operacional.

Segurança do operador

O outro grande contribuinte para eventos operacionais que introduzem falhas são as pessoas. Humanos cometem erros. Eles podem usar as credenciais erradas, digitar o comando errado, pressionar Enter muito cedo ou perder uma etapa crítica. Realizar ações manuais de forma consistente resulta em erro, resultando em falha.

Uma das principais causas de erros do operador são interfaces de usuário confusas, não intuitivas ou inconsistentes. Jim Gray também observou em seu estudo de 1985 que “as interfaces que pedem informações ao operador ou que ele execute alguma função devem ser simples, consistentes e tolerantes a falhas do operador”. (Veja Jim Gray, "Por que os computadores param e o que pode ser feito a respeito? “, junho de 1985, Relatório Técnico Tandem 85.7.) Essa percepção continua sendo verdadeira hoje. Nas últimas três décadas, existem vários exemplos em todo o setor em que uma interface de usuário confusa ou complexa, a falta de confirmação ou instruções ou até mesmo uma linguagem humana hostil fizeram com que um operador fizesse a coisa errada.

> Regra 12
>
> Faça com que seja fácil para os operadores fazerem a coisa certa.

Prevenir sobrecarga

O último colaborador comum de impacto são seus clientes, os usuários reais de sua workload. Workloads bem-sucedidas tendem a ser muito usadas, mas às vezes esse uso supera a capacidade de escalabilidade da workload. Muitas coisas podem acontecer: os discos podem ficar cheios, os pools de threads podem se esgotar, a largura de banda da rede pode ficar saturada ou os limites de conexão do banco de dados podem ser atingidos.

Não há um método à prova de falhas para eliminá-las, mas o monitoramento proativo da capacidade e da utilização por meio de métricas da Operational Health fornecerá avisos antecipados quando essas falhas puderem ocorrer. Técnicas como redução de carga, disjuntores e repetição com recuo exponencial e instabilidade podem ajudar a minimizar o impacto e aumentar a taxa de sucesso, mas essas situações ainda representam falha. O escalonamento automatizado com base nas métricas da Operational Health pode ajudar a reduzir a frequência de falhas devido à sobrecarga, mas pode não ser capaz de responder com rapidez suficiente às mudanças na utilização.

Se você precisar garantir a capacidade continuamente disponível para os clientes, precisará fazer concessões quanto à disponibilidade e ao custo. Uma forma de garantir que a falta de capacidade não leve à indisponibilidade é fornecer uma cota a cada cliente e garantir que a capacidade de sua workload seja escalada para fornecer 100% das cotas alocadas. Quando os clientes excedem sua cota, eles são limitados, o que não é uma falha e não conta para a disponibilidade. Você também precisará acompanhar de perto sua base de clientes e prever a utilização futura para manter a capacidade suficiente provisionada. Isso garante que sua workload não seja direcionada a cenários de falha devido ao consumo excessivo de seus clientes.

- Amazon Builders' Library — Usando a redução de carga para evitar sobrecarga
- Amazon Builders' Library — Imparcialidade em sistemas multilocatários

Por exemplo, vamos examinar uma workload que fornece um serviço de armazenamento. Cada servidor na workload pode suportar 100 downloads por segundo, os clientes recebem uma cota de 200 downloads por segundo e há 500 clientes. Para poder suportar esse volume de clientes, o serviço precisa fornecer capacidade para 100.000 downloads por segundo, o que requer 1.000 servidores. Se algum cliente exceder sua cota, ele será limitado, o que garante capacidade suficiente para todos os outros clientes. Este é um exemplo simples de uma forma de evitar a sobrecarga sem rejeitar unidades de trabalho.

Conclusão

Estabelecemos 12 regras para alta disponibilidade em todo este documento.

- Regra 1 — Falhas menos frequentes (MTBF mais longo), tempos de detecção de falhas mais curtos (MTTD mais curto) e tempos de reparo mais curtos (MTTR mais curto) são os três fatores usados para melhorar a disponibilidade em sistemas distribuídos.
- Regra 2 — A disponibilidade do software em sua workload é um fator importante da disponibilidade geral de sua workload e deve receber o mesmo foco que outros componentes.
- Regra 3 — Reduzir as dependências pode ter um impacto positivo na disponibilidade.
- Regra 4 — Em geral, selecione dependências cujas metas de disponibilidade sejam iguais ou maiores que as metas da sua workload.
- Regra 5 — Use a economia para aumentar a disponibilidade das dependências em uma workload.
- Regra 6 — Há um limite superior para a eficiência de custos de poupar. Utilize o mínimo de peças de reposição necessárias para alcançar a disponibilidade necessária.
- Regra 7 — Não use dependências em ambientes de gerenciamento em seu plano de dados, especialmente durante a recuperação.
- Regra 8 — Acople as dependências de forma flexível para que sua workload possa operar corretamente apesar do comprometimento da dependência, sempre que possível.
- Regra 9 — A observabilidade e a instrumentação são fundamentais para reduzir o MTTD e o MTTR.
- Regra 10 — Concentre-se na mitigação do impacto, não na resolução de problemas. Escolha o caminho mais rápido de volta à operação normal.
- Regra 11 — O isolamento de falhas diminui o escopo do impacto e aumenta o MTBF da workload ao reduzir a taxa geral de falhas.
- Regra 12 — Faça com que seja fácil para os operadores fazerem a coisa certa.

A melhoria da disponibilidade da workload é impulsionada pela redução do MTTD e do MTTR e pelo aumento do MTBF. Em resumo, discutimos as seguintes formas de melhorar a disponibilidade que abrangem tecnologia, pessoas e processos.

- MTTD
  - Reduza o MTTD por meio do monitoramento proativo de suas métricas de experiência do cliente.

  - Aproveite as verificações de integridade granulares para um failover rápido.
- MTTR
  - Monitore o escopo do impacto e as métricas de saúde operacional.
  - Reduza o MTTR seguindo 1/Restart, 2/Reboot, 3/Reimage/Redeploy e 4/Replace.
  - Contorne o fracasso compreendendo o escopo do impacto.
  - Utilize serviços que tenham tempos de reinicialização mais rápidos, como contêineres e funções sem servidor em máquinas virtuais ou hosts físicos.
  - Reverta automaticamente implantações com falha quando possível.
  - Estabeleça runbooks e ferramentas operacionais para operações de diagnóstico e procedimentos de reinicialização.
- MTBF
  - Elimine bugs e defeitos no software por meio de testes rigorosos antes de serem lançados para produção.
  - Implemente engenharia de caos e injeção de falhas.
  - Utilize a quantidade certa de economia nas dependências para tolerar falhas.
  - Minimize o escopo do impacto durante falhas por meio de contêineres de falhas.
  - Implemente padrões para implantações e mudanças.
  - Projete interfaces de operação simples, intuitivas, consistentes e bem documentadas.
  - Estabeleça metas para a excelência operacional.
  - Favoreça a estabilidade em vez do lançamento de novos recursos quando a disponibilidade é uma dimensão crítica de sua workload.
  - Implemente cotas de uso com limitação ou redução de carga, ou ambas, para evitar sobrecarga.

Lembre-se de que nunca seremos totalmente bem-sucedidos na prevenção de falhas. Concentre-se em projetos de software com o melhor isolamento possível de falhas que limita o escopo e a magnitude do impacto, de preferência mantendo esse impacto abaixo dos limites de “tempo de inatividade” E invista em detecção e mitigação muito rápidas e confiáveis. Os sistemas distribuídos modernos ainda precisam encarar o fracasso como inevitável e ser projetados em todos os níveis para oferecer alta disponibilidade.

Apêndice 1 — Métricas críticas de MTTD e MTTR

A seguir está uma estrutura para padronização em instrumentação e observabilidade que pode ajudar a reduzir o MTTD e o MTTR durante um evento.

Métricas de experiência do cliente. Essas métricas refletem que um serviço é responsivo e está disponível para atender às solicitações dos clientes. Por exemplo, latência do ambiente de gerenciamento. Essas métricas medem a taxa de erro, a disponibilidade, a latência, o volume e a taxa de aceleração.

Métricas de avaliação de impacto. Essas métricas fornecem informações sobre o escopo do impacto durante os eventos. Por exemplo, o número ou a porcentagem de clientes afetados por um evento do plano de dados. Mede o número ou a porcentagem de coisas afetadas.

Métricas operacionais de saúde. Essas métricas refletem que um serviço é responsivo e está disponível para atender às solicitações dos clientes, mas se concentra em subsistemas e recursos de infraestrutura comuns. Por exemplo, a porcentagem de utilização da CPU de sua frota de EC2. Essas métricas devem medir a utilização, a capacidade, a taxa de transferência, a taxa de erro, a disponibilidade e a latência.

Colaboradores

Os colaboradores deste documento incluem:

- Michael Haken, arquiteto de soluções principal, Amazon Web Services

Outras fontes de leitura

Para obter informações adicionais, consulte:

- Pilar de confiabilidade Well-Architected
- Pilar Excelência operacional da Well-Architected
- Amazon Builders' Library — Garantindo a segurança de reversão durante implantações
- Amazon Builders' Library — Beyond five 9s: lições de nossos planos de dados mais altos disponíveis
- Amazon Builders' Library — Automatização de implantações seguras e sem intervenção
- Amazon Builders' Library — Arquitetando e operando sistemas resilientes sem servidor em grande escala
- Amazon Builders' Library — A abordagem da Amazon para implantação de alta disponibilidade
- Amazon Builders' Library — A abordagem da Amazon para criar serviços resilientes
- Amazon Builders' Library — A abordagem da Amazon para falhar com sucesso
- Centro de Arquitetura do AWS

Histórico do documentos

Para ser notificado sobre atualizações desse whitepaper, inscreva-se no feed RSS.

Alteração	Descrição	Data
Publicação inicial	Whitepaper publicado pela primeira vez.	12 de novembro de 2021

> **Note**
>
> Para assinar as atualizações de RSS, você deve ter um plug-in de RSS habilitado para o navegador que você está usando.

Avisos

Os clientes são responsáveis por fazer sua própria avaliação independente das informações contidas neste documento. Este documento: (a) é apenas para fins informativos, (b) representa as ofertas e práticas de produtos atuais da AWS, que estão sujeitas a alterações sem aviso prévio e (c) não criam nenhum compromisso ou garantia da AWS e de suas afiliadas, fornecedores ou licenciadores. Os produtos ou serviços da AWS são fornecidos “no estado em que se encontram”, sem garantias, representações ou condições de qualquer tipo, expressas ou implícitas. As responsabilidades e as obrigações da AWS com os seus clientes são controladas por contratos da AWS, e este documento não é parte, nem modifica, qualquer contrato entre a AWS e seus clientes.

Glossário do AWS

Para obter a terminologia mais recente da AWS, consulte o glossário da AWS na Referência do Glossário da AWS.