Como implementar verificações de integridade

David Yanacek

É possível projetar serviços com todo tipo de confiabilidade e resiliência. No entanto, para serem confiáveis na prática, eles também precisam saber lidar com falhas previsíveis. Na Amazon, criamos serviços horizontalmente escalonáveis e redundantes, porque o hardware é projetado para eventualmente ter falhas. Todo disco rígido tem uma vida útil máxima, assim como qualquer software é suscetível ter falhas em algum momento. A integridade de um servidor pode parecer binária: ou ela funciona, ou não funciona e ele acaba sendo desativado. Infelizmente, isso não é verdade. Acreditamos que servidores com falha não são apenas desativados, mas podem causar problemas imprevisíveis e às vezes desproporcionais ao sistema. As verificações de integridade podem detectar e reagir a problemas desse tipo automaticamente.

Este artigo descreve como usar verificações de integridade para detectar e lidar com falhas de servidor único, além de apresentar o que acontece quando as verificações de integridade não são realizadas e como os sistemas que reagem exageradamente a falhas na verificação de integridade podem transformar pequenos problemas em interrupções totais. Também oferecemos reflexões sobre nossa experiência na Amazon no balanceamento de prós e contras de diferentes tipos de implementação de verificações de integridade.

Falhas pequenas com impactos enormes

Quando comecei meu trabalho como desenvolvedor de software na Amazon, eu fazia parte da equipe de renderização do site Amazon.com. Enquanto eu trabalhava para implementar alguns instrumentos e ter uma visão melhor sobre o funcionamento do software, infelizmente inseri um bug. O bug ocorria raramente. No entanto, quando isso acontecia, ele fazia um determinado servidor da web renderizar páginas de erro em branco para todas as solicitações. A única forma de corrigir esse problema era reiniciar o servidor web. Detectamos o bug e revertemos a alteração rapidamente, adicionamos diversos testes e melhoramos os processos para identificar casos assim no futuro. Mas, enquanto o bug estava na produção, alguns servidores de uma frota grande ficaram nesse estado defeituoso.

Uma coisa que tornou o bug especialmente difícil de encontrar foi o fato de o servidor não perceber que não estava íntegro. Além disso, o servidor perdeu a habilidade de informar a integridade aos sistemas de monitoramento. Por isso, ele não for retirado de serviço automaticamente e não ativou os alarmes regulares. Para piorar ainda mais, o servidor ficou *muito* rápido e começou a gerar páginas de erro em branco a uma velocidade muito maior em comparação à velocidade com que outros "servidores íntegros" geravam páginas corretas. A tecnologia de balanceamento de carga usada naquele momento favoreceu servidores rápidos em detrimento dos lentos, então ela direcionou um volume desproporcional de tráfego para os servidores não íntegros, o que aumentou ainda mais o impacto.

Como o monitoramento envolve a medição de taxas de erro e latência de vários pontos do sistema, outros alarmes foram ativados. Embora esses tipos de sistemas de monitoramento e processos operacionais possam servir como uma proteção para conter o problema, verificações de integridade corretas podem minimizar significativamente o impacto de erros assim por meio da detecção e reação rápida a falhas.

Prós e contras de verificações de integridade

Verificações de integridade são uma maneira de perguntar a um serviço de um servidor específico se ele é capaz de realizar o trabalho com êxito ou não. Load balancers fazem essa pergunta a todos os servidores periodicamente a fim de definir para quais deles é seguro direcionar o tráfego. Um serviço que sonda mensagens de uma fila pode perguntar *a si mesmo* se etá íntegro antes de decidir sondar mais trabalho da fila. Agentes de monitoramento (que podem estar em execução em cada servidor

ou em uma frota de monitoramento externa) perguntam as servidores se eles estão íntegros para definir se podem emitir um alarme ou lidar automaticamente com os servidores com falha.

Como vimos no meu exemplo do bug do site, quando um servidor não íntegro continua em serviço, ele pode reduzir a disponibilidade do serviço como um todo de maneira desproporcional. Em uma frota de dez servidores, um servidor com problema pode significar que a disponibilidade da frota seria de 90% ou menos. O que torna isso ainda pior é que alguns algoritmos de balanceamento de carga, como o de "menor número de solicitações", direcionam mais trabalho ao servidor mais rápido. Quando há uma falha em um servidor, ele começa a rejeitar as solicitações rapidamente e a atrair mais solicitações do que servidores íntegros, criando um "buraco negro" na frota de serviço. Em alguns casos, adicionamos uma proteção extra para evitar buracos negros: reduzir a velocidade de solicitações rejeitadas para que ela corresponda à média de latência de solicitações bem-sucedidas. No entanto, há outros contextos, como no caso de instrumentos de sondagem de filas, em que o problema é mais difícil de contornar. Por exemplo, se um instrumento de sondagem de filas estiver sondando mensagens na mesma velocidade em que ele é capaz de recebê-las, uma falha no servidor também causará um buraco negro. Devido à diversidade do conjunto de ambientes de distribuição de trabalhos, a nossa forma de pensar sobre a proteção de um servidor com falha parcial varia de sistema para sistema.

Servidores falham de maneira *independente* por uma série de razões, incluindo discos que se tornam ingraváveis e fazem as solicitações falharem imediatamente, relógios que sofrem alterações abruptas e causam erros na autenticação de chamadas para as dependências, servidores que não conseguem recuperar materiais criptografados atualizados e geram falhas na descriptografia e criptografia, processos críticos de suporte que falham devido a bugs próprios, vazamentos de memória e dependências que congelam processos.

Servidores também falham devido a motivos *correlacionados* que fazem com que vários ou todos os servidores de uma frota falhem ao mesmo tempo. Motivos correlacionados incluem quedas de energia de uma dependência compartilhada e problemas de rede de grande escala.

Uma verificação de integridade ideal testa todos os aspectos da integridade do servidor e do aplicativo, talvez até verificando se os processos de suporte não críticos estão em execução. No entanto, os problemas surgem quando uma verificação de integridade falha por causa de um motivo *não crítico* e quando uma falha está *correlacionada* a vários servidores. Se a automação remove os servidores de um serviço em situações em que eles ainda poderiam ser úteis, essa automação traz mais problemas do que soluções.

A parte mais difícil de verificações de integridade é a tensão entre, de um lado, os benefícios de realizar verificações de integridade minuciosas e migrar falhas de servidores únicos rapidamente e, do outro lado, as complicações causadas em toda a frota por falhas relacionadas a falsos-positivos. Assim, um dos desafios de criar uma boa verificação de integridade é a proteção cuidadosa contra falsos-positivos. Em geral, isso significa que a automação de verificações de integridade deve interromper o direcionamento de tráfego a um único servidor com falha, mas manter o tráfego caso a frota *pareça* estar com problemas.

Maneiras de mensurar a integridade

Várias partes de um servidor podem ter problemas, e há vários pontos do sistema em que medimos a integridade do servidor. Algumas verificações de integridade são capazes de indicar que um servidor específico teve um problema independente, enquanto outros são mais vagos e indicam falsos-

positivos no caso de falhas correlacionadas. Algumas verificações de integridade são difíceis de implementar. Outras são implementadas no momento da configuração, por exemplo, em serviços como Amazon Elastic Compute Cloud (Amazon EC2) e Elastic Load Balancing. Cada tipo de verificação de integridade tem pontos fortes.

Verificações de vivacidade

Verificações de vivacidade testam a conectividade básica de um serviço e a presença de um processo do servidor. Elas costumam ser realizadas por um load balancer ou agente de monitoramento externo e não consideram os detalhes de funcionamento de aplicativos. Verificações de vivacidade costumam ser incluídas com o serviço e não exigem implementações adicionais pelo autor do aplicativo. Alguns exemplos de verificações de vivacidade que usamos na Amazon incluem:

- Testes que confirmam que um servidor está monitorando a porta esperada e aceitando novas conexões TCP.
- Testes que realizam solicitações HTTP básicas e garantem que o servidor responda com um código de status 200.
- Verificações de status para o Amazon EC2 que testam elementos básicos necessários para o funcionamento de qualquer sistema, como acessibilidade de rede.

Verificações locais de integridade

Verificações locais de integridade vão além de verificações de vivacidade e avaliam a probabilidade de o aplicativo funcionar. Essas verificações de integridade testam recursos locais que não são compartilhados com os pares do servidor. Por isso, elas têm pouca probabilidade de falhar em vários servidores da frota simultaneamente. Essas verificações de integridade testam:

- A inabilidade de gravar ou ler do disco. Pode ser tentador acreditar que um serviço stateless não precisa de um disco gravável. No entanto, os serviços da Amazon costumam usar discos para ações como monitoramento, registro em log e publicação de dados assíncronos de medição.
- Falhas ou problemas em processos críticos. Alguns serviços aceitam solicitações usando um proxy do servidor (similar ao NGINX) e executam a lógica do negócio em outros processo do servidor. Às vezes, verificações de vivacidades só testam se o processo do proxy está sendo executado. Um processo de verificação local de integridade pode passar do proxy para o aplicativo e verificar se os dois estão sendo executando e atendendo às solicitações corretamente. É interessante observar que, no exemplo do site apresentado no início do artigo, a verificação de integridade existente era profunda o bastante para garantir que o processo de renderização estava sendo executado e respondendo, mas não para garantir que estava respondendo corretamente.
- Processos de suporte ausentes: hosts sem daemons de monitoramento podem deixar operadores sem informações sobre a integridade dos serviços. Outros processos de suporte enviam registros de medição e faturamento de uso ou recebem atualizações sobre as credenciais. Servidores com processos de suporte falhos colocam a funcionalidade em risco de maneiras sutis e difíceis de detectar.

Verificações de integridade de dependência

Verificações de integridade de dependência são uma inspeção minuciosa da capacidade de um aplicativo interagir com os sistemas adjacentes. O ideal é que essas verificações apontem problemas locais (como credenciais expiradas) que estejam impedindo o servidor de interagir com uma dependência. Mesmo

assim, elas também podem incluir falsos-positivos quando houver problemas com a própria dependência. Devido a esses falsos-positivos, precisamos ter cuidado ao reagir a falhas nas verificações de integridade de dependências. Verificações de integridade de dependências podem testar:

- Configurações ruins ou metadados obsoletos: se um processo sondar por atualizações nos metadados ou na configuração de maneira assíncrona e o mecanismo de atualização estiver com falha em um servidor, o servidor pode ficar significativamente fora de sincronia em relação aos pares e agir de maneira imprevisível e não testada. No entanto, se um servidor passar algum tempo sem atualizações, ele não sabe se o mecanismo de atualização está com falha ou se o sistema de atualização central interrompeu a atualização de todos os servidores.
- A incapacidade de se comunicar com outros servidores ou dependências: comportamentos estranhos na rede afetam a capacidade de subconjuntos de servidores se comunicarem com dependências sem afetar o envio de tráfego ao servidor. Problemas de software, como bloqueios ou bugs de pools de conexão, também podem prejudicar a comunicação da rede.
- Outros bugs de software pouco comuns que causam a devolução de um processo: bloqueios, vazamentos de memória e bugs de corrupção de estado podem causar erros no servidor.

Detecção de anomalias

A detecção de anomalias analisa todos os servidores de uma frota e determina se algum deles está se comportando de maneira estranha em comparação aos pares. Graças à agregação de dados de monitoramento por servidor, podemos comparar continuamente taxas de erro, dados de latências e outros atributos para identificar servidores anômalos e desativá-los automaticamente. A detecção de anomalias aponta divergências na frota que não podem ser identificadas por um servidor, como:

- Erro de relógio: especialmente quando servidores estão com uma carga alta, os relógios podem ser alterados de maneira abrupta e drástica. Medidas de segurança, como as usadas para avaliar solicitações assinadas da AWS, permitem que o horário mostrado pelo relógio de um cliente esteja com apenas cinco minutos de diferença em relação ao horário real. Caso contrário, as solicitações feitas a produtos da AWS falham.
- Código antigo: se um servidor estava desconectado da rede ou ficou desligado por muito tempo para depois ser reativado, pode ser que ele esteja executando um código perigoso obsoleto e incompatível com o restante da frota.
- Qualquer modo de falha inesperado: às vezes, servidores falham e acabam retornando erros identificados como sendo do cliente, e não dos próprios servidores (HTTP 400 em vez de 500). Em vez de uma falha, a velocidade dos servidores pode ser reduzida ou eles podem responder mais rápido que os pares, o que é um sinal de que estão retornando falsos-positivos aos chamadores. A detecção de anomalias é uma forma incrível de detectar falhas inesperadas.

Para a detecção de anomalias funcionar na prática, alguns pressupostos precisam ser cumpridos:

- Os servidores precisam estar fazendo aproximadamente a mesma coisa: nos casos em que roteamos diferentes tipos de tráfego para diferentes tipos de servidores, pode ser que os servidores não se comportem de maneira similar o bastante para detectar anomalias. No

entanto, quando usamos load balancers para direcionar o tráfego aos servidores, é provável que eles respondam de maneiras semelhantes.
- As frotas precisam ser relativamente homogêneas: em frotas com diferentes tipos de instâncias, algumas delas podem ser mais lentas que outras, o que pode acionar uma detecção passiva falsa de servidor com falha. Para contornar esse cenário, agrupamos métricas por tipo de instância.
- Os erros ou a diferença de comportamento precisam ser relatados: como confiamos que os próprios servidores relatarão erros, o que acontece quando os sistemas de monitoramento deles também estão com falha? Felizmente, o cliente de um serviço é um ótimo lugar para adicionar instrumentação. Load balancers como o Application Load Balancer publicam logs que mostram qual servidor de back-end foi contactado em cada solicitação, o tempo de resposta e se a solicitação teve êxito ou não.

Como reagir de maneira segura a falhas de verificação de integridade

Quando um servidor identifica que está não íntegro, ele pode executar duas ações. No caso mais extremo, ele decide internamente que não deve receber trabalho e se desativa falhando uma verificação de integridade de um load balancer ou parando de sondar uma fila. O servidor também pode reagir informando uma autoridade central sobre o problema e permitindo que o sistema central determine como lidar com a questão. O sistema central pode lidar com o problema de maneira segura sem permitir que a automação derrube toda a frota.

Há várias formar de implementar e reagir a verificações de integridade. Esta seção descreve alguns padrões usados na Amazon.

Modo de falha aberta

Alguns load balancers podem agir como uma autoridade central inteligente. Quando um servidor individual falha em uma verificação de integridade, o load balancer para de enviar tráfego para ele. Mas, quando todos os servidores falham simultaneamente nas verificações de integridade, o load balancer entra em modo de falha aberta e encaminha tráfego a todos os servidores. Podemos usar load balancers para auxiliar na implementação segura de uma verificação de integridade de dependências, talvez incluindo uma que consulte o banco de dados e as verificações para garantir que os processos de suporte não críticos estejam funcionando.

Por exemplo, o AWS Network Load Balancer entra em modo de falha aberta se nenhum servidor estiver indicando integridade. Ele também sai de zonas de disponibilidade não íntegras se todos os servidores de uma zona de disponibilidade indicarem que não estão íntegros. (Para obter mais informações sobre como usar o Network Load Balancers para realizar verificações de integridade, consulte a documentação sobre o Elastic Load Balancing.) Nosso Application Load Balancer também é compatível com o modo de falha aberta, assim como o Amazon Route 53. (Para obter mais informações sobre como configurar verificações de integridade com o Route 53, consulte a documentação sobre o Route 53.)

Quando confiamos no modo de falha aberta, não deixamos de testar os modos de falha da verificação de integridade de dependências. Por exemplo, imagine um serviço em que os servidores se conectam a um armazenamento de dados compartilhados. Se o armazenamento de dados ficar devagar ou responder com uma taxa baixa de erros, os servidores podem ocasionalmente falhar nas verificações de integridade de dependências. Essa condição faz com que os servidores alternem entre o funcionamento e a interrupção de serviços, mas sem acionar o modo de falha aberta. É importante ponderar e testar falhas parciais de dependências nessas verificações de integridade para evitar casos em que uma falha possa fazer com as verificações profundas de integridade piorem a situação.

Embora entrar no modo de falha aberta seja uma ação útil, na Amazon, costumamos ser céticos em relação a coisas que não podemos ponderar ou testar em todas as situações. Ainda não temos provas de o modo de falha aberta é acionado conforme esperamos em todos os tipos de sobrecarga, falhas parciais ou "falhas cinzas" ("gray failure") de um sistema ou suas dependências. Por causa dessa limitação, as equipes da Amazon tendem a só permitir a ação rápida de load balancers em verificações locais de integridade. Para os outros casos, elas confiam que os sistemas centralizados reagirão de maneira cuidadosa a verificações de integridade de dependências mais profundas. Isso não significa que não usamos o modo de falha aberta nem comprova que ele funciona em casos específicos. Mas, quando é possível que uma lógica seja aplicada a um grande número de servidores rapidamente, tratamos essa lógica com cautela.

Verificações de integridade sem disjuntores

Pode parecer que a forma mais rápida e simples de recuperar um sistema é permitindo que os servidores reajam aos próprios problemas. No entanto, essa também é a maneira mais arriscada caso o servidor esteja errado em relação à integridade ou ao contexto geral da frota. Quando todos os servidores da frota tomam a mesma decisão simultaneamente, isso pode resultar em falhas consecutivas em todos os serviços adjacentes. Esse risco tem um benefício. Se houver uma lacuna no monitoramento e na verificação de integridade, um dos servidores poderá reduzir a disponibilidade de um serviço até que o problema seja detectado. No entanto, esse cenário evita que um comportamento inesperado na verificação de integridade de toda a frota cause uma queda total do serviço.

Estas são as práticas recomendadas para a implementação de verificações de integridade sem um disjuntor integrado:

- Configurar o produtor do trabalho (load balancer, thread de sondagem de fila) para realizar verificações locais de *vivacidade* e *integridade*. Os servidores só são retirados do serviço automaticamente pelo load balancer se tiverem algum problema que seja definitivamente restrito ao servidor, como um disco com falha.
- Configurar outros sistemas de monitoramento externos para realizar *verificações de integridade de dependências* e *detectar anomalias*. Esses sistemas podem tentar interromper instâncias automaticamente ou ainda engajar ou emitir alarmes para um operador.

Durante a criação de sistemas que reagem automaticamente a falhas na verificação de integridade de dependências, é necessário estabelecer um limite adequado para evitar que o sistema automatizado tome medidas drásticas inesperadamente. As equipes da Amazon que operam servidores stateful, como Amazon DynamoDB, Amazon S3 e Amazon Relational Database Service (Amazon RDS), têm exigências importantes de durabilidade ligadas à substituição de servidores. Elas também criam limitações cautelosas de taxas e controlam os ciclos de feedback para interromper a automação e envolver humanos quando os limites são ultrapassados. Quando criamos automações assim, é necessário garantir que as falhas de verificação de integridade de dependências sejam notadas. Para algumas métricas, é necessário que os servidores enviem um relatório sobre o status deles a um sistema de monitoramento central. Para compensar casos em que o servidor está com tantas falhas que é incapaz de enviar relatórios de integridade, você precisa sondá-lo ativamente para verificar a integridade.

Priorizar a integridade

Especialmente em condições de sobrecarga, é importante que os servidores priorizem as verificações de integridade em vez do trabalho normal. Nessa situação, falhar ou demorar para responder a verificações de integridade pode piorar ainda mais uma situação de baixa disponibilidade.

Quando um servidor falha uma verificação de integridade de um load balancer, ele está pedindo para o load balancer retirá-lo do serviço imediatamente e por um período considerável. Uma falha de servidor único não é um problema. No entanto, quando há uma onda de tráfego no serviço, a última coisa que queremos é reduzir o tamanho do serviço. Retirar os servidores dos serviços durante uma sobrecarga pode ser um poço sem fundo. Forçar os outros servidores a assumir ainda mais tráfego deixa-os mais suscetíveis a sobrecargas, falhas na verificação de integridade e redução ainda maior da frota.

O problema não é os servidores sobrecarregados retornarem erros. Mas sim os servidores não responderem à solicitação de ping do load balancer a tempo. Afinal de contas, as verificações de integridade do load balancer são configuradas com limites de tempo, assim como qualquer outra chamada de serviço remota. Servidores com baixa disponibilidade demoram para responder por uma série de razões, incluindo alta contenção de CPU, ciclos longos de coleta de lixo ou a simples falta de threads de operadores. Os serviços precisam ser configurados para ignorar os recursos e responder a verificações de integridade de maneira pontual, sem assumir solicitações demais.

Felizmente, há algumas práticas recomendadas de configuração que seguimos para ajudar a evitar esses poços sem fundo. Ferramentas como o iptables e até mesmo alguns load balancers são compatíveis com a noção de “número máximo de conexões”. Nesse caso, o sistema operacional (ou o load balancer) limita o número de conexões com o servidor para que o processo não seja sobrecarregado por solicitações concorrentes que causariam uma redução na velocidade.

Quando um serviço conta com um proxy ou um load balancer compatível com um número máximo de conexões, parece lógico que o número de threads de operadores do servidor HTTP seja igual ao número máximo de conexões do proxy. No entanto, em caso de indisponibilidade, essa configuração faria com que o serviço entrasse em um poço sem fundo. As verificações de integridade do proxy também precisam de conexões. Por isso, é importante que o pool de operadores do servidor seja grande o bastante para acomodar solicitações extras de verificação de integridade. Operadores ociosos são baratos, então tendemos a configurar operadores extras: podem ser só alguns ou até o dobro do número máximo de conexões do proxy.

Outra estratégia para priorizar verificações de integridade é fazer com que os servidores restrinjam por si mesmo o número máximo de solicitações concorrentes. Nesse caso, as verificações de integridade do load balancer são sempre permitidas, mas solicitações normais são rejeitadas caso o servidor já esteja no limite. Na Amazon, as implementações variam de um semáforo no Java até um análise mais complexa de tendências de uso de CPUs.

Outra forma de ajudar a garantir que os serviços respondam a tempo a uma solicitação de ping de verificação de integridade é aplicando a lógica de verificação de integridade de dependências em um thread de segundo plano e atualizando um marcador isHealthy que seja verificado pela lógica do ping. Nesse caso, os servidores respondem prontamente a verificações de integridade e a verificação de integridade de dependências gera uma carga previsível no sistema externo com que ela interage. Quando as equipes fazem isso, elas têm um cuidado redobrado para detectar um falha no thread de verificação de integridade. Caso esse thread de segundo plano saia, o servidor não detectará uma falha (ou recuperação!) futura no servidor.

Como equilibrar verificações de integridade com o escopo do impacto

Verificações de integridade de dependências são interessantes porque atuam como um teste minucioso da integridade de um servidor. Infelizmente, elas podem ser perigosas porque uma dependências pode causar uma série de falhas sequenciais em todo o sistema.

Para entender melhor como lidar com dependências de verificações de integridade, podemos analisar a arquitetura de serviços utilizada na Amazon. Todo serviço da Amazon é projetado para fazer uma série de coisas: não há um serviço monolítico que faça tudo. Há várias razões para preferirmos projetar serviços assim, inclusive o aumento da velocidade de inovação em equipes pequenas e a redução do escopo do impacto em caso de problema em algum serviço. Esse projeto de arquitetura também pode ser aplicado a verificações de integridade.

Quando um serviço chama outro, ele assume uma dependência nesse serviço. Se um serviço só chamar essa dependência às vezes, podemos considerá-la uma "dependência leve", porque o serviço pode continuar realizando alguns trabalho mesmo se não puder se comunicar com a dependência. Quando uma verificação de integridade testa uma dependência sem a proteção do modo de falha aberta, essa dependência é considerada "forte". Se essa dependência estiver inoperante, o serviço também é interrompido, criando uma série de falhas com um escopo de impacto cada vez maior.

Mesmo que as funcionalidades sejam separadas em diferentes serviços, cada um deles provavelmente atenderá a várias APIs. Às vezes, APIs têm dependências próprias. Se uma API for afetada, é preferível que o serviço continue atendendo às outras APIs. Por exemplo, um serviço pode ser tanto um plano de controle (como APIs CRUD chamadas ocasionalmente em recursos duradouros) quanto um plano de dados (APIs supercríticas para o negócio e com alta taxa de transferência). Gostaríamos que as APIs do plano de dados continuassem funcionando mesmo se as APIs do plano de controle tivessem problemas para se comunicar com as suas dependências.

Similarmente, até mesmo uma única API pode se comportar de maneira diferente dependendo da entrada ou do estado dos dados. Um padrão comum é uma API Read que consulta um banco de dados mas mantém as respostas em um cachê local por um tempo. Se o banco de dados estiver inoperante, o serviço ainda pode usar as leituras em cachê até o banco de dados voltar a funcionar. Se as verificações de integridade falharem quando houver apenas um caminho de código não íntegro, o escopo do impacto de um problema de comunicação com uma dependência será maior.

Essa discussão sobre quais dependências devem passar pela verificação de integridade levanta uma questão interessante sobre os prós e contras de microsserviços e serviços relativamente monolíticos. Raramente há uma regra clara sobre em quantas unidades implantáveis ou endpoints um serviço deve ser dividido, mas as perguntas sobre "quais dependências devem passar pela verificação de integridade" e sobre se "uma falha aumenta o escopo do impacto" são perspectivas interessantes para ajudar a determinar se um serviço deve ser micro ou macro.

Problemas reais que ocorreram durante verificações de integridade

Tudo isso faz sentido em teoria, mas o que acontece na prática quando a verificação da integridade de sistemas não é realizada corretamente? Procuramos histórias recorrentes de clientes da AWS e da própria Amazon para ajudar a ilustrar a visão geral. Também procuramos por fatores de

compensação, as diversas ações das equipes para evitar que um deslize em uma verificação de integridade cause um problema mais abrangente.

Implantações

Um dos problemas recorrentes de verificações de integridade envolve implantações. Sistemas de implantação, como o AWS CodeDeploy, enviam novos códigos por push a um subconjunto da frota por vez, aguardando uma onda de implantações ser concluída antes de passar para a próxima. Para que esse processo funcione, os servidores precisam se reportar ao sistema de implantação quando estiverem executando o novo código. Se eles não reportarem, o sistema de implantação considera que há algo errado com o novo código e reverte a implantação.

O script de implantação de inicialização de serviço mais básico simplesmente dividiria o processo do servidor e responderia imediatamente "implantação concluída" ao sistema de implantação. No entanto, isso é perigoso porque várias coisas podem dar errado com o novo código: ele poderia falhar logo após ser iniciado, ser desligado e não conseguir escutar um soquete de servidor, não ser capaz de carregar a configuração necessária para processar solicitações ou encontrar um bug. Quando um sistema de implantação não está configurado para executar o teste de verificação de integridade, ele não percebe que está enviando uma implantação problemática. Ele persiste e causa falhas nos servidores, um a um.

Felizmente, na prática, as equipes da Amazon implementam vários sistemas de mitigação para evitar que esse cenário torne toda a frota inoperante. Um exemplo de mitigação é a configuração de alarmes que são acionados sempre que o tamanho geral da frota esteja pequeno demais ou executando uma carga alta, ou ainda quando houver uma alta taxa de latência ou de erros. Se algum desses alarmes é acionado, o sistema interrompe e reverte a implantação.

Outro tipo de mitigação é o uso de implantações em fases. Em vez de incluir toda a frota em uma única implantação, o serviço pode ser configurado para implantar um subconjunto, talvez uma zona de disponibilidade, antes de pausar e executar um conjunto completo de testes na zona. Esse processo de implantação por zona de disponibilidade é prático porque os serviços já são criados para operar caso haja problemas em uma única zona de disponibilidade.

E é claro que, antes de implantar para a produção, as equipes da Amazon executam essas mudanças em ambientes de teste e realizam testes automatizados de integração que identificariam falhas desse tipo. No entanto, podem haver diferenças sutis e inevitáveis entre os ambientes de teste e produção. Por isso, é importante combinar vários níveis da implantação de segurança para identificar problemas antes que eles afetem a produção. Embora verificações de integridade sejam importantes para evitar más implantações, isso não é tudo que fazemos. Nós pensamos sobre as abordagens alternativas que servem como bloqueios para proteger frotas contra erros desse tipo e outros.

Processadores assíncronos

Outra falha recorrente está relacionada ao processamento assíncrono de mensagens, como um serviço que recebe trabalho sondando uma fila SQS ou o Amazon Kinesis Stream. Diferente de sistemas que recebem solicitações de load balancers, não há nada executando verificações de integridade automaticamente para desativar servidores.

Quando os serviços não têm verificações de integridade que sejam profundas o suficiente, servidores com operadores de filas individuais podem ter falhas, como discos cheios ou sem descritores de arquivos. Esse problema não faz com que o servidor pare de puxar trabalhos da fila, mas ele perde a capacidade de

processar mensagens. Esse problema já causou atrasos no processamento de mensagens, com o servidor problemático extraindo trabalhos da fila rapidamente sem conseguir resolvê-los.

Nesse tipo de situação, normalmente há vários fatores de compensação para ajudar a reduzir o impacto. Por exemplo, se um servidor não conseguir processar a mensagem puxada do SQS, o SQS envia a mensagem para outro servidor após um tempo limite de visibilidade de mensagem configurado anteriormente. A latência de ponta a ponta aumenta, mas as mensagens não são abandonadas. Outro fator de compensação é um alarme que é ativado sempre que há um número alto demais de mensagens de erro de processamento, alertando um operador a investigar.

Preenchimento de discos

Outra classe de falhas é o preenchimento de discos de servidores, o que causa falhas no processamento e no registro de logs. Essa falha gera uma lacuna na visibilidade do monitoramento, porque pode ser que o servidor não consiga relatar as falhas para o sistema de monitoramento.

Mais uma vez, vários controles de mitigação evitam que os serviços tomem decisões por si mesmas e rapidamente mitigam o impacto. As taxas de erro e métricas de latência de sistemas com proxy, como Application Load Balancer ou API Gateway, serão geradas pelo proxy. Nesse caso, os alarmes são ativados mesmo se o servidor não as estiver relatando. Para sistemas baseados em filas, serviços como Amazon Simple Queue Service (Amazon SQS) relatam as métricas que indicam atrasos no processamento de algumas mensagens.

O que essas soluções têm em comum são os vários níveis de monitoramento. Os erros são relatados pelo servidor, mas também por um sistema externo. Em verificações de integridade, esse princípio também é importante. Um sistema externo pode testar a integridade de um determinado sistema com mais precisão do que é capaz de testar a si mesmo. É por isso que as equipes usam o AWS Auto Scaling para configurar um load balancer para realizar verificações de integridade externa por ping.

As equipes também criam um sistema personalizado de verificação de integridade para averiguar a integridade de cada servidor e relatar ao AWS Auto Scaling caso algum deles não esteja íntegro. Uma implementação comum desse sistema envolve uma função do Lambda que é executada a cada minuto para testar a integridade de todos os servidores. Essas verificações de integridade podem até salvar o estado entre cada execução em algo como o DynamoDB, para evitar que vários servidores sejam marcados como não íntegros de uma só vez sem nenhum tipo de aviso.

Zumbis

Outro problema recorrente são os servidores zumbis. Os servidores podem ser desconectados da rede por um tempo e continuar sendo executados, ou podem ser desligados por períodos longos e depois reinicializados.

Quando servidores zumbis são reativados, eles podem estar consideravelmente dessincronizados em relação ao restante da frota, o que pode causar problemas sérios. Por exemplo, se um servidor zumbi estiver executando uma versão muito antiga e incompatível de um software, ele pode usar uma configuração errada ou causar falhas quando interagir com um banco de dados que tenha um esquema diferente.

Para lidar com zumbis, os sistemas costumam responder verificações de integridade com a versão do software que estão executando no momento. Depois disso, um agente de monitoramento central compara as respostas de toda a frota para verificar se algum servidor está executando uma versão obsoleta e evitando que ele volte para o serviço.

Em conclusão

Servidores e os softwares que eles executam sofrem falhas por vários motivos estranhos. O hardware tem uma vida útil. Como desenvolvedores de software, nós acabamos escrevendo bugs como aquele que descrevi acima, o que causa falhas no software. Para identificar todos os tipos de falhas, são necessários vários níveis de verificação, desde verificações leves de vivacidade até o monitoramento passivo de métricas por servidor.

Quando as falhas ocorrem, é importante detectá-las e rapidamente retirar os servidores ativados do serviço. No entanto, assim como com qualquer automação de frota, nós adicionamos limitações de taxas e disjuntores que desativam a automação e envolvem humanos caso surjam incertezas ou situações extremas. A falha aberta e a construção de atores centralizados são estratégias que aproveitam as verificações de integridade profundas sem abrir mão da segurança da automação com limitações de taxas.