# Garantindo a segurança da reversão durante as implantações

**Sandeep Pokkunuri**

Um dos princípios orientadores de como criamos soluções na Amazon é: evite atravessar portas de mão única. Isso significa que evitamos escolhas difíceis de reverter ou ampliar. Aplicamos este princípio em todas as etapas do desenvolvimento do software — do design dos produtos, recursos, APIs e sistemas de back-end até as implantações. Neste artigo, descreverei como aplicamos esse princípio às implantações de software.

Uma implantação leva um ambiente de software de um estado (versão) para outro. O software pode funcionar perfeitamente bem em qualquer um desses estados. No entanto, o software pode não funcionar bem durante ou após a transição para frente (atualização ou rolar para frente) ou transição para trás (downgrade ou reversão). Quando o software não funciona bem, isso leva a uma interrupção do serviço tornando-o não confiável para os clientes. Neste artigo, presumo que as versões do software estejam funcionando conforme o esperado. Meu foco é como garantir que rolar para frente ou para trás durante a implantação não leve a erros.

Antes de lançar uma nova versão do software, testamos em um ambiente de teste beta ou gama em várias dimensões, como funcionalidade, simultaneidade, desempenho, escala e tratamento posterior das falhas. Esse teste nos ajuda a descobrir quaisquer problemas na nova versão e corrigi-los. No entanto, nem sempre é suficiente garantir uma implantação bem-sucedida. Podemos nos deparar com circunstâncias inesperadas ou comportamento abaixo do ideal do software em ambientes de produção. Na Amazon, queremos evitar nos colocar em uma situação em que reverter a implantação possa causar erros para nossos clientes. Para evitar essa situação, nos preparamos totalmente para uma reversão antes de cada implantação. Uma versão do software que pode ser revertida sem erros nem interrupções da funcionalidade disponível na versão anterior é chamada de compatível com versões anteriores. Planejamos e verificamos se nosso software é compatível com versões anteriores a cada revisão.

Antes de entrar em detalhes sobre como a Amazon aborda as atualizações de software, vamos discutir algumas das diferenças entre implantações de software distribuídas e autônomas.

# Implantações de software autônomas vs. distribuídas

Para o software autônomo executado como um processo único em um dispositivo, as implantações são atômicas. Duas versões do software nunca são executadas ao mesmo tempo. Se o software autônomo mantiver o estado, a nova versão deverá ler (ou seja, desserializar) os dados gravados (ou seja, serializados) pela versão antiga e vice-versa. A satisfação dessa condição torna a implantação segura para rolar para frente e para trás.

Em um sistema distribuído, as implantações são mais complexas. As implantações são feitas através de atualizações contínuas, para que a disponibilidade não seja afetada. A nova versão é lançada em um subconjunto de hosts de uma só vez, para que os outros hosts possam continuar a atender as solicitações. Normalmente, esses hosts se comunicam através de uma chamada de procedimento remoto (RPC, do inglês Remote Procedure Call) ou de um estado persistente compartilhado (por exemplo, metadados ou pontos de verificação). Tal comunicação ou estado compartilhado pode apresentar desafios adicionais. O gravador e o leitor podem estar executando versões diferentes do software. Como consequência, eles podem interpretar os dados de maneira diferente. O leitor pode até não conseguir ler os dados completamente, causando uma interrupção.

# Problemas com alterações de protocolo

Constatamos que a razão mais comum para não conseguir uma reversão é uma alteração de protocolo. Por exemplo, considere uma alteração de código que comece a compactar dados

gravando-os no disco. Depois que a nova versão gravar alguns dados compactados, a reversão não será mais uma opção. A versão antiga não sabe que deve descompactar os dados após a leitura do disco. Se os dados estiverem armazenados em um blob ou em um armazenamento de documentos, sua leitura por outros servidores falhará, mesmo que a implantação esteja em andamento. Se esses dados passarem entre dois processos ou servidores, o receptor não conseguirá lê-los.

Às vezes, as alterações no protocolo podem ser muito sutis. Por exemplo, considere dois servidores que se comunicam de forma assíncrona através de uma conexão. Para se manterem cientes de que estão ativos, eles concordam em enviar uma pulsação a cada cinco segundos. Se um servidor não vir uma pulsação dentro do tempo estipulado, ele presume que o outro servidor está inoperante e encerra a conexão.

Agora, considere uma implantação que aumente o período de pulsação para 10 segundos. A confirmação do código parece menor — apenas uma alteração de número. Contudo, agora não é seguro rolar para frente e para trás. Durante a implantação, o servidor que está executando a nova versão envia uma pulsação a cada 10 segundos. Consequentemente, o servidor que está executando a versão antiga não vê pulsação por mais de cinco segundos e encerra a conexão com o servidor que está executando a nova versão. Em uma frota grande, essa situação pode ocorrer com várias conexões, levando a uma queda na disponibilidade.

É difícil analisar essas alterações sutis através da leitura de códigos ou de documentos de design. Portanto, verificamos explicitamente se cada implantação é segura para rolar para frente e para trás.

# Técnica de implantação em duas fases

Uma maneira de garantir que possamos reverter com segurança é usar uma técnica comumente chamada de implantação em duas fases. Considere o seguinte cenário hipotético com um serviço que gerencia dados (gravações e leituras) no Amazon Simple Storage Service (Amazon S3). O serviço é executado em uma frota de servidores em várias zonas de disponibilidade para escalabilidade e disponibilidade.

Atualmente, o serviço usa o formato XML para a manutenção dos dados. Como mostrado no diagrama a seguir na versão V1, todos os servidores gravam e leem XML. Por razões comerciais, queremos manter os dados no formato JSON. Se fizermos essa alteração em uma implantação, os servidores que detectaram a alteração gravarão em JSON. Mas, os outros servidores ainda não sabem ler JSON. Esta situação provoca erros. Portanto, dividimos tal alteração em duas partes e executamos uma implantação em duas fases.


Como mostrado no diagrama anterior, chamamos a primeira fase de Preparar. Nesta fase, preparamos todos os servidores para ler JSON (além de XML), mas eles continuam gravando XML enquanto implantam a versão V2. Essa alteração não muda nada do ponto de vista operacional. Todos os servidores ainda podem ler XML e todos os dados ainda são gravados em XML. Se

decidirmos reverter essa alteração, os servidores reverterão para uma condição na qual não poderão ler JSON. Isso não é um problema, porque nenhum dos dados foi gravado em JSON ainda.

Como mostrado no diagrama anterior, chamamos a segunda fase de Ativar. Nesta fase, ativamos os servidores para usar o formato JSON para gravação, implantando a versão V3. À medida que cada servidor detectar essa alteração, ele começa a escrever em JSON. Os servidores que ainda precisam detectar essa alteração ainda podem ler JSON porque foram preparados na primeira fase. Se decidirmos reverter essa alteração, todos os dados gravados pelos servidores que estavam temporariamente na fase Ativar estarão em JSON. Os dados gravados por servidores que não estavam na fase Ativar estão em XML. Essa situação é satisfatória porque, como mostrado na V2, os servidores ainda podem ler XML e JSON após a reversão.
Embora o diagrama anterior mostre a alteração do formato de serialização de XML para JSON, a técnica geral é aplicável a todas as situações descritas anteriormente na seção Alterações de protocolo. Por exemplo, lembre-se do cenário anterior em que o período de pulsação entre servidores teve que ser aumentado de cinco para 10 segundos. Na fase Preparar, podemos fazer com que todos os servidores reduzam o período de pulsação esperado para 10 segundos, embora todos os servidores continuem enviando uma pulsação a cada cinco segundos. Na fase Ativar, alteramos a frequência para uma vez a cada 10 segundos.

## Precauções com implantações em duas fases

Agora, descreverei as precauções que tomamos ao seguir a técnica de implantação em duas fases. Embora eu me refira ao cenário de exemplo descrito na seção anterior, essas precauções se aplicam à maioria das implantações em duas fases.

Várias ferramentas de implantação permitem que os usuários considerem uma implantação bem-sucedida se um número mínimo de hosts detectar a alteração e se declarar como íntegro. Por exemplo, AWS CodeDeploy tem uma configuração de implantação chamada minimumHealthyHosts.

Uma suposição crítica na implantação em duas fases do exemplo é que, no final da primeira fase, todos os servidores foram atualizados para ler XML e JSON. Se um ou mais servidores não conseguirem atualizar durante a primeira fase, eles não poderão ler os dados durante e após a segunda fase. Portanto, verificamos explicitamente que todos os servidores detectaram a alteração na fase Preparar.

Quando eu estava trabalhando no Amazon DynamoDB, decidimos alterar o protocolo de comunicação entre um grande número de servidores abrangendo vários microsserviços. Eu coordenava as implantações entre todos os microsserviços para que todos os servidores alcançassem a fase Preparar primeiro e depois continuassem para a fase Ativar. Como precaução, verifiquei explicitamente que a implantação foi bem-sucedida em todos os servidores no final de cada fase.

Embora cada uma das duas fases seja segura para reverter, não podemos reverter as duas alterações. No exemplo anterior, no final da fase Ativar, os servidores gravam dados em JSON. A versão do software em uso antes das alterações Preparar e Ativar não sabe ler JSON. Assim, como precaução, deixamos passar um período considerável de tempo entre as fases Preparar e Ativar. Chamamos esse período de período de bake, e sua duração é geralmente de alguns dias. Esperamos para ter certeza de que não precisamos reverter para uma versão anterior.

Após a fase Ativar, não podemos remover com segurança a capacidade do software de ler XML. A remoção não é segura porque todos os dados gravados antes da fase Preparar estão em XML. Só podemos remover sua capacidade de ler XML depois de garantir que todos os objetos tenham sido regravados em JSON. Chamamos este processo de preenchimento. Ele pode exigir ferramentas adicionais que podem ser executadas simultaneamente enquanto o serviço está gravando e lendo os dados.

# Melhores práticas para serialização

A maioria dos softwares envolve serialização de dados — seja por persistência ou transferência pela rede. À medida que evolui, é comum que a lógica de serialização mude. As alterações podem variar de adicionar um novo campo até alterar completamente o formato. Ao longo dos anos, chegamos a algumas melhores práticas que seguimos para a serialização:

- Geralmente, evitamos o desenvolvimento de formatos de serialização personalizados.

  A lógica inicial para serialização personalizada pode parecer trivial e até fornecer melhor desempenho. Contudo, iterações subsequentes do formato apresentam desafios que já foram resolvidos por estruturas bem estabelecidas, como JSON, Protocol Buffers, Cap'n Proto e FlatBuffers. Quando usadas adequadamente, essas estruturas fornecem recursos de segurança, como saída, compatibilidade com versões anteriores e rastreamento de existência de atributos (ou seja, se um campo foi definido explicitamente ou se foi atribuído implicitamente um valor padrão).

- A cada alteração, atribuímos explicitamente uma versão distinta aos serializadores.

  Fazemos isso independentemente do código fonte ou da criação da versão. Também armazenamos a versão do serializador com os dados serializados ou nos metadados. Versões mais antigas do serializador continuam funcionando no novo software. Consideramos geralmente útil emitir uma métrica para a versão dos dados gravados ou lidos. Se houver erros, ela fornece aos operadores informações de visibilidade e solução de problemas. Tudo isso é aplicado também às versões RPC e API.
- Evitamos serializar estruturas de dados que não podemos controlar.

  Por exemplo, podemos serializar os objetos da coleção do Java usando reflexão. Mas, ao tentarmos atualizar o JDK, a implementação subjacente de tais classes pode ser alterada, causando falha na desserialização. Esse risco também é aplicado a classes de bibliotecas compartilhadas entre equipes.
- Normalmente, projetamos serializadores para permitir a presença de atributos desconhecidos.

  Quando possível, nossos serializadores retêm atributos desconhecidos enquanto gravam os dados. Com esta acomodação, mesmo se um servidor que estiver executando a nova versão do software incluir novos atributos nos dados durante a serialização, os servidores que executam a versão antiga não limparão os atributos ao atualizar os mesmos dados. Portanto, não é necessária uma implantação em duas fases.

Como em muitas de nossas melhores práticas, nós as compartilhamos com cuidado porque nossas diretrizes não se aplicam a todos os aplicativos e cenários.

# Verificação da segurança de reversão para uma alteração

Geralmente, verificamos explicitamente se uma alteração de software é segura para rolar para frente e para trás através do que chamamos de teste de atualização e downgrade. Para esse processo, configuramos um ambiente de teste representativo dos ambientes de produção. Ao longo dos anos, identificamos alguns padrões que evitamos durante a configuração de ambientes de teste.

Vi situações em que a implantação de uma alteração na produção provocou erros, embora ela tenha passado em todos os testes no ambiente de teste. Em uma ocasião, os serviços no ambiente de teste tinham apenas um servidor cada. Assim, todas as implantações eram atômicas, o que impedia a possibilidade de executar diferentes versões do software simultaneamente. Agora, mesmo que os ambientes de teste não tenham tanto tráfego quanto os ambientes de produção, usamos vários servidores de diferentes zonas de disponibilidade por trás de cada serviço, exatamente como seria na produção. Na Amazon, adoramos a frugalidade, mas não quando se trata de garantir a qualidade.

Em outra ocasião, o ambiente de teste tinha vários servidores. No entanto, a implantação foi realizada em todos os servidores de uma só vez para acelerar o teste. Essa abordagem também impediu a execução de versões antigas e novas do software ao mesmo tempo. Problemas para rolar para frente não foram detectados. Agora usamos a mesma configuração de implantação em todos os ambientes de teste e produção.

Para as alterações que envolvem coordenação entre microsserviços, mantemos a mesma ordem de implantação nos microsserviços nos ambientes de teste e produção. No entanto, a ordem para rolar para frente e para trás pode ser diferente. Por exemplo, geralmente seguimos uma ordem específica no contexto da serialização. Ou seja, os leitores vêm antes dos gravadores quando rolam para frente e os gravadores vêm antes dos leitores quando rolam para trás. A ordem apropriada é comumente seguida em ambientes de teste e produção.

Quando uma configuração do ambiente de teste é semelhante aos ambientes de produção, simulamos o tráfego de produção o mais próximo possível. Por exemplo, criamos e lemos vários registros (ou mensagens) em sucessão rápida. Todas as APIs são exercitadas continuamente. Em seguida, assumimos o ambiente em três etapas, cada uma com duração razoável para identificar possíveis erros. A duração é longa o suficiente para que todas as APIs, fluxos de trabalho de back-end e tarefas em lote sejam executadas pelo menos uma vez.
Primeiro, implantamos a alteração em aproximadamente metade da frota para garantir a coexistência da versão do software. Em segundo lugar, concluímos a implantação. E por último, iniciamos a implantação de reversão e seguimos as mesmas etapas até que todos os servidores executem o software antigo. Se não houver erros ou comportamento inesperado durante essa fases, consideramos o teste bem-sucedido.

# Conclusão

Para tornar um serviço confiável é essencial poder reverter uma implantação sem interrupções para nossos clientes. Testar explicitamente a segurança da reversão elimina a necessidade de confiar na análise manual, que pode estar sujeita a erros. Quando descobrimos que uma alteração não é segura para reversão, normalmente podemos dividi-la em duas alterações, sendo que cada uma delas é segura para rolar para frente e para trás.