Como acelerar com a entrega contínua

Mark Mansour

Melhoria contínua e automação de software

Há mais de 10 anos, realizamos um projeto na Amazon para entender com que rapidez nossas equipes estavam transformando ideias em sistemas de produção de alta qualidade. Isso nos levou a medir o throughput de software para que pudéssemos melhorar a velocidade de nossa execução. Descobrimos que estava levando, em média, 16 dias do check-in do código à produção. Na Amazon, as equipes começavam com uma ideia e depois levavam normalmente um dia e meio para escrever um código e dar vida à ideia. Levávamos menos de uma hora para compilar e implantar o novo código. O restante do tempo, quase 14 dias, passávamos esperando que os membros da equipe iniciassem uma compilação, fizessem as implantações e executassem testes. No final do projeto, recomendávamos automatizar nossos processos pós-check-in para melhorar nossa velocidade de execução. O objetivo era eliminar atrasos, mantendo ou até melhorando a qualidade.

No centro dessa recomendação havia um programa de melhoria contínua para aumentar a velocidade de nossa execução. Baseamos nossa decisão de melhorar nossa velocidade de execução no princípio de liderança da Insistência nos mais altos padrões. Esse princípio se trata de ter padrões incansavelmente altos, elevá-los continuamente e fornecer produtos, serviços e processos de alta qualidade. Nossos Princípios de Liderança descrevem como a Amazon faz negócios, como os líderes lideram e como mantemos o cliente no centro de nossas decisões.

A Amazon já havia criado ferramentas de desenvolvimento de software para tornar nossos engenheiros de software mais produtivos. Criamos nosso próprio sistema de compilação centralizado e hospedado, o Brazil, que executa uma série de comandos em um servidor com o objetivo de gerar um artefato que possa ser implantado. Nesse ponto, o Brazil não detectava mudanças no código-fonte. Uma pessoa tinha que iniciar uma compilação. Também tínhamos nosso próprio sistema de implantação, o Apollo, que exigia o upload de um artefato de compilação como parte do início de uma implantação. Inspirados pelo interesse na entrega contínua que ocorre no setor, criamos nosso próprio sistema, o Pipelines, para automatizar o processo de entrega de software entre o Brazil e o Apollo.

Pipelines: nossa ferramenta de implantação contínua

Iniciamos um programa piloto para automatizar o processo de entrega de software para um pequeno número de equipes. Quando terminamos, a equipe piloto principal havia alcançado uma redução de 90% no tempo total necessário para ir do check-in à produção.

O projeto validou o conceito de pipeline como uma maneira de nossas equipes definirem todas as etapas necessárias para liberar o software para os clientes. A primeira etapa em um pipeline é compilar um artefato. Em seguida, o pipeline executa esse artefato compilado por meio de uma série de etapas até que o artefato seja liberado para todos os clientes. Usamos pipelines para reduzir o risco de que uma nova alteração de código possa ter um impacto negativo em nossos clientes. Cada etapa do pipeline deve aumentar nossa confiança de que o artefato compilado não contenha defeitos. Se um defeito atingir a produção, queremos que ela volte a um estado íntegro o mais rápido possível.

Quando lançamos o Pipelines, ele só podia modelar um único processo de liberação por aplicativo. Essa limitação gerou consistência, padronização e simplificação dos processos de liberação de uma equipe. Isso resultou em menos defeitos. Antes de passarmos a usar pipelines, era comum que as equipes tivessem diferentes processos de lançamento para correções de bugs e lançamentos de recursos importantes. À medida que outras equipes viram o sucesso das equipes que haviam executado o piloto de entrega automatizada, começaram a migrar os processos de liberação gerenciados delas manualmente para os pipelines, para que também pudessem melhorar a consistência. As equipes que costumavam ter uma variedade de processos de liberação agora tinham um processo padronizado que todos usavam. Além disso, quando elas mudaram os processos de liberação delas para uma ferramenta, os membros da equipe frequentemente revisitavam a abordagem deles e encontravam maneiras de simplificar o processo.

A equipe do Pipelines tinha metas anuais para aumentar o uso com a "adoção sedutora". Em outras palavras, eles precisavam tornar o produto tão bom que as pessoas exigissem usá-lo. Medimos o número de equipes que usam um pipeline para implantar o software delas na produção e classificamos os pipelines de acordo com o nível de automação. Vimos equipes atingindo metas de usar um pipeline para liberar software e avançar para liberações totalmente automatizadas. No entanto, percebemos que, em algumas organizações, a maneira como avaliamos a qualidade pode levar as equipes a automatizar o processo de liberação delas sem fornecer nenhum teste.

A resposta à pergunta "quanto teste é suficiente?" é uma decisão difícil. Ela exige que uma equipe entenda o contexto em que opera. Para lidar com essa situação, usamos outro princípio de liderança: Propriedade. Este princípio se trata de pensar em longo prazo e não sacrificar o valor em longo prazo para obter resultados em curto prazo. As equipes de software da Amazon têm um alto nível de teste e dedicam muito esforço a isso, porque ser proprietário de um produto significa também ser proprietário das consequências de quaisquer defeitos nesse produto. Se um problema vier a ter impacto sobre os clientes, são os membros da pequena equipe de software de thread única que lidam com esse problema e o corrigem em tempo real. A tensão entre aumentar a velocidade da execução e responder a problemas na produção significa que as equipes são motivadas a testar adequadamente. No entanto, se investirmos demais em testes, talvez não tenhamos sucesso porque outros se moveram mais rápido que nós. Estamos sempre buscando melhorar nossos processos de lançamento de software sem nos tornarmos um obstáculo para os negócios.

Outro problema que enfrentamos é que as equipes não estavam aprendendo as melhores práticas de lançamento de software entre si. As equipes de thread única são incentivadas a trabalhar de forma autônoma, e isso significava que os engenheiros estavam resolvendo seus problemas de implantação de forma independente. Quando encontrassem uma solução que atendesse às suas necessidades de lançamento de software, promoveriam a técnica para outros engenheiros por meio de listas de discussão, reuniões de operações e outros canais de comunicação. Havia dois problemas com esse estilo de comunicação. Primeiro, esses canais são um canal de comunicação de melhor esforço, o que significa que nem todos aprenderam sobre as novas técnicas. Segundo, os líderes que incentivavam suas equipes a adotar a nova melhor prática não tinham como entender se suas equipes haviam feito o trabalho necessário para realmente adotar a melhor prática. Percebemos que precisávamos ajudar todos os engenheiros

a terem acesso às melhores práticas que aprendemos e oferecer aos líderes a capacidade de identificar os pipelines que precisavam de atenção.

Nossa solução foi mecanizar nosso aprendizado adicionando verificações de melhores práticas nas ferramentas que usamos para criar e liberar software. Éramos sensíveis ao fato de que uma melhor prática para uma organização pode não ser uma melhor prática para outra, portanto, permitimos que essas verificações fossem configuradas por organização. As verificações de melhores práticas em nível organizacional deram aos líderes a capacidade de adaptar os processos de liberação deles para atender às necessidades de negócios deles. As verificações de melhores práticas em nível organizacional deram aos líderes a capacidade de adaptar os processos de liberação deles para atender às necessidades dos negócios deles. Colocar mensagens nas ferramentas quase garantia que os membros da equipe aprendessem sobre as melhores práticas e quando elas entrassem em vigor. Descobrimos que, dando às equipes tempo para aprender e debater sobre uma nova melhor prática, uma organização teve a chance de repetir e melhorar as verificações dela. Em última análise, isso levou a melhorar a qualidade das melhores práticas e a melhor adesão da comunidade de engenharia.

Identificamos as melhores práticas a aplicar sistematicamente. Um grupo de nossos engenheiros mais experientes catalogou as razões comuns para um lançamento não funcionar. Eles identificaram as etapas que teriam feito o lançamento funcionar. Em seguida, usávamos essa lista para criar nosso conjunto de verificações de melhores práticas. Com esse processo percebemos que, embora desejássemos que novas revisões de software chegassem aos clientes instantaneamente, sem esforço e sem degradar a disponibilidade, priorizamos a disponibilidade primeiro, depois a velocidade e, em seguida, facilitamos para nossos engenheiros.

Reduzindo o risco de que um defeito atinja os clientes

Espera-se que todos os engenheiros, em algum momento, introduzam um defeito em um de nossos sistemas. Nossos processos de liberação, incluindo nossos pipelines e sistemas de implantação, devem ser projetados para identificar esses defeitos o mais rápido possível e impedir que eles causem impacto em nossos clientes. Precisamos garantir que nossos processos de liberação estejam configurados corretamente e que nosso artefato compilado funcione conforme o planejado.

Higiene da implantação: a maneira mais básica de testar a implantação garante que o artefato recém-implantado possa ser iniciado e responder ao trabalho. Como parte do fluxo de trabalho pós-implantação, executamos verificações rápidas que garantem que o artefato recém-implantado tenha iniciado e esteja atendendo ao tráfego. Por exemplo, usamos ganchos de eventos do ciclo de vida no arquivo AppSpec do AWS CodeDeploy para acionar scripts simples com a finalidade de parar, iniciar e validar a implantação. Também verificamos se temos capacidade suficiente para atender ao tráfego de clientes. Criamos técnicas como mínimo de hosts íntegros no CodeDeploy para validar que sempre temos capacidade suficiente para atender nossos clientes. Por fim, se o mecanismo de implantação puder detectar uma falha, ele deverá reverter a alteração para minimizar o tempo em que os clientes veem um defeito.

Testes antes da produção: uma das melhores práticas da Amazon é automatizar os testes de unidade, integração e pré-produção e adicionar esses testes ao nosso pipeline. Insistimos em realizar testes de carga e segurança, com a tendência de adicionar esses testes aos nossos pipelines. Quando dizemos testes de unidade, queremos dizer todos os testes que você pode querer executar em sua máquina de compilação, incluindo verificações de estilo, cobertura de código, complexidade de código e muito mais. Pensamos nos testes de integração como incluindo todos os testes prontos para uso, como injeção de falha, teste automatizado de navegador e similares. Há muitos artigos excelentes sobre testes de unidade e integração, então não vou entrar em mais detalhes aqui.

Nosso teste de unidade e integração visa verificar se o comportamento do nosso artefato compilado está funcionalmente correto. Quanto mais validação for realizada, menor o risco de um defeito ser exposto aos nossos clientes. Para reduzir o tempo necessário para colocar um produto nas mãos de nossos clientes, tentamos detectar um defeito o mais cedo possível no processo de liberação. Em geral, isso significa que, se seus testes forem menores e mais rápidos, você receberá um feedback mais rápido sobre quaisquer problemas com suas alterações.

Na Amazon, também usamos uma técnica que chamamos de *teste pré-produção*. Um ambiente de pré-produção é o último local em que o teste ocorre antes de implantarmos nossas alterações na produção. Um teste do ambiente de pré-produção usa a configuração de produção do sistema para que ele atue exatamente como um sistema de produção. Essa abordagem tem dois benefícios. O primeiro é que os ambientes de pré-produção testam a configuração de produção para garantir que o serviço possa se conectar corretamente a todos os recursos de produção, incluindo datastores de produção. Segundo, garante que o sistema interaja corretamente com as APIs dos serviços de produção dos quais depende. Os ambientes de pré-produção são usados apenas pela equipe proprietária desse serviço e nunca recebem tráfego de clientes. A execução de testes de pré-produção aumenta nossa confiança de que os mesmos código e configuração funcionarão na produção.

Validação na produção: quando liberamos código para nossos clientes, não fazemos tudo de uma vez. O escopo do impacto de liberar um defeito para todos os clientes ao mesmo tempo é muito grande. Em vez disso, implantamos em células: uma instância de um serviço completamente independente. Quando implantamos alterações em nosso primeiro conjunto de clientes em nossa primeira célula, somos extremamente cautelosos. Permitimos que apenas um pequeno número de clientes veja a nova alteração e obtemos feedback sobre o funcionamento correto do novo código. Monitoramos a quantidade de erros que nossos serviços emitem após uma implantação canário. Se a taxa de erro aumentar, reverteremos a alteração automaticamente. Por exemplo, podemos esperar 3.000 pontos de dados positivos, sem pontos negativos, antes de continuar a implantação.

Pode surgir uma complicação se os seus testes automatizados perderem um caso de uso. Nós nos esforçamos para detectar todos os erros com nossos testes estruturados e repetíveis, sejam eles automáticos ou manuais. No entanto, mesmo quando nos esforçamos ao máximo, um defeito pode passar despercebido. Para fazer nossos testes, deixamos a nova alteração na produção por um período fixo para ver se um membro que não faz parte da equipe encontrará um problema. Passamos muito tempo debatendo se devemos deixar as alterações pararem na produção ou quanto tempo devemos esperar após uma implantação canário antes de implantar

no restante do grupo de implantação. Muitas de nossas equipes decidiram esperar por um período fixo, além de reunir pontos de dados positivos antes de prosseguir em nossa rotina de implantação. A quantidade de tempo que um pipeline espera depende muito da equipe. Algumas equipes esperam horas e outras esperam minutos. Quanto maior o impacto e quanto mais tempo para corrigir o problema, mais lento será o processo de liberação.

Depois de ganharmos confiança na primeira célula, exporemos a nova alteração de código a mais e mais clientes até que seja completamente liberada. Assim como fizemos com a implantação canário, esperamos ganhar confiança na implantação da primeira nova célula antes de avançar para a próxima célula. À medida que ganhamos mais confiança no artefato compilado, reduzimos o tempo gasto verificando a alteração do código. Isso resulta em um padrão no qual pretendemos ir do check-in ao nosso primeiro cliente de produção o mais rápido possível. No entanto, após a produção, liberamos lentamente o novo código para os clientes, trabalhando para ganhar mais confiança à medida que aceleramos o restante de nossas implantações.

Para garantir que nossos sistemas de produção continuem atendendo às solicitações dos clientes, geramos tráfego sintético em nossos sistemas. Queremos um feedback rápido se nosso serviço não estiver funcionando corretamente, por isso executamos nossos testes sintéticos pelo menos a cada minuto. Projetamos testes sintéticos para garantir que nossos processos em execução estejam íntegros e que todas as dependências sejam testadas, o que geralmente envolve testar todas as APIs voltadas ao público.

Controle de quando o software é lançado: para controlar a segurança de nossos lançamentos de software, criamos mecanismos que nos permitem controlar a velocidade com que as mudanças passam pelo nosso pipeline. Usamos métricas, janelas de tempo e verificações de segurança para controlar quando nosso software é lançado.

Os pipelines podem ser configurados para impedir uma implantação quando um alarme é disparado com base em uma alteração nas métricas. Usamos métricas generalizadas e temos alarmes sobre a integridade de nossos sistemas, a integridade de nossas células, zonas e regiões de disponibilidade e quase tudo o que você pode imaginar. Configuramos nossos pipelines para interromper a implantação do código quando uma métrica importante dispara um alarme. No entanto, às vezes uma equipe precisa implantar uma correção para que o alarme do sistema seja resolvido. Nesse cenário, permitimos que as equipes cancelassem os alarmes, impedindo que as mudanças se movessem por um pipeline.

Nossos pipelines podem especificar uma janela de tempo em que as alterações podem progredir por um pipeline. As equipes podem encontrar suas próprias janelas de tempo para restringir quando as mudanças chegam aos clientes. As equipes da AWS preferem lançar o software quando há muitas pessoas que podem responder rapidamente e atenuar um problema causado por uma implantação. Para tornar isso realidade, as equipes geralmente definem suas janelas de tempo para que sejam implantadas apenas durante o horário comercial. Outras equipes da Amazon preferem lançar software quando há pouco tráfego de clientes. Essas janelas de tempo podem ser substituídas, se necessário.

Também temos a capacidade de interromper um pipeline com base no conteúdo do artefato compilado. Por exemplo, podemos bloquear um artefato compilado que contenha um pacote inválido conhecido ou uma referência específica ao Git. Usamos esse recurso quando descobrimos que uma alteração em um pacote continha uma regressão de performance. Se apenas removêssemos o pacote de nosso repositório de pacotes, os pipelines que já continham o pacote com defeito ainda implantariam essa alteração ruim nos clientes.

Como abordamos a velocidade de nossa execução

Descobrimos que as equipes estão ansiosas para adotar a automação. Estamos todos muito motivados a criar e liberar recursos para os clientes que melhoram as vidas deles, e a entrega contínua torna isso sustentável. Vimos a automação retroceder aos engenheiros, removendo trabalhos manuais frustrantes, propensos a erros e trabalhosos. Mostramos que a implantação contínua tem um impacto positivo na qualidade. Vimos que a automação permite que as equipes liberem com frequência, uma alteração de cada vez, facilitando a identificação de regressões.

Quando os sistemas são novos, a área de superfície a ser testada geralmente é compreensível pela maioria dos membros da equipe, o que torna alguns testes manuais tratáveis. No entanto, à medida que os sistemas se tornam mais complexos e os membros da equipe mudam, o valor da automação aumenta. Gostamos de automatizar nossos sistemas para que possamos nos concentrar em agregar valor ao cliente, em vez de gerenciar manualmente o processo de obter essas mudanças para os clientes.

Por muitos anos, a Amazon executa programas de melhoria contínua focados na velocidade com a qual liberamos software para os clientes e na segurança dessas versões. Não começamos com todas as verificações e testes de risco sobre os quais escrevi neste artigo. Com o tempo, inventamos maneiras de identificar e mitigar riscos.

Os programas de melhoria contínua são executados por líderes de negócios em diferentes níveis da organização. Isso permite que cada líder de negócios ajuste seu processo de liberação de software para corresponder aos riscos e ao impacto nos negócios deles. Alguns de nossos programas de melhoria contínua são executados em grandes seções da Amazon e, às vezes, líderes de organizações menores executam seus próprios programas. Sabemos que sempre há exceções à regra. Nossos sistemas têm mecanismos de exclusão, para não desacelerar as equipes que precisam de uma isenção permanente ou temporária. Por fim, nossas equipes são proprietárias do comportamento do software delas e são responsáveis por investir adequadamente no processo de liberação de software.

Começamos medindo onde estavam nossos pontos fracos, abordando-os e iterando. Para tornar esse trabalho sustentável, precisávamos fazê-lo de forma incremental e celebrar as melhorias ao longo do tempo. Quando a Amazon começou a usar pipelines, muitas equipes não tinham certeza de que a implantação contínua funcionaria. Para fazer as equipes começarem, as incentivamos a codificar o processo de liberação atual, etapas manuais e tudo, em um pipeline. Para muitas equipes, o pipeline delas agia como uma interface visual para o processo de liberação sem promover automaticamente artefatos compilados por meio de um processo de liberação. À medida que aumentava a confiança, elas ativaram gradualmente a automação

em diferentes estágios do pipeline até que não precisassem mais acionar manualmente nenhuma etapa do pipeline.

Avanço rápido para hoje. Agora, a Amazon está no ponto em que as equipes buscam a automação completa enquanto escrevem um novo código. Para nós, a automação é a única maneira de continuarmos a expandir nossos negócios.