Automatização de implantações seguras sem intervenção manual

Clare Liguori

Automatização de implantações seguras sem intervenção manual

Quando fui entrevistada para meu cargo na Amazon, fiz questão de perguntar a um dos entrevistadores: “Com que frequência você implanta na produção?” Na época, eu estava trabalhando em um produto em que uma versão principal era desenvolvida uma ou duas vezes por ano, mas algumas vezes eu precisava lançar uma pequena correção entre as grandes versões. Cada correção lançada precisava de horas da minha total atenção para ser desenvolvida. Em seguida, eu verificava freneticamente os logs e as métrica para ver se eu havia cometido algum erro após a implantação e precisava voltar ao desenvolvimento.

Eu li que a Amazon praticava a implantação contínua, então, quando fui entrevistada, queria saber quanto tempo eu teria que passar gerenciando e observando implantações como uma desenvolvedora na Amazon. O entrevistador me disse que as alterações eram implantadas automaticamente na produção várias vezes por dia por pipelines de implantação contínua. Quando perguntei quanto tempo do dia ele passava dedicando total atenção a cada uma dessas implantações e observando logs e métrica sem busca de algum impacto como eu vinha fazendo, ele disse “normalmente nenhum”. Como os pipelines faziam esse trabalho para a equipe dele, a maioria das implantações não era observada ativamente por ninguém. “Uau!” Eu disse. Depois que eu ingressei na Amazon, estava empolgada para descobrir exatamente como essas implantações automatizadas “sem intervenção manual” funcionavam.

Como a implantação contínua e segura libera o tempo dos desenvolvedores

Desde então, tenho visto em primeira mão como a Amazon configura os pipelines de implantação contínua para nos ajudar a implantar com rapidez e segurança. Passei a apreciar a maneira como nossas práticas de segurança para implantação contínua liberam o desenvolvedor do tempo usado para trabalhar nas implantações. Quando envio o código de produção para a ramificação principal do repositório de códigos fonte do meu serviço, normalmente não me preocupo mais com esse código e passo para a minha próxima tarefa enquanto o pipeline da minha equipe assume a implantação dessa alteração na produção. A liberação da minha alteração de código para um serviço de produção é totalmente automatizada pelo pipeline, o que significa que a última interação ou revisão de uma parte do código feita por mim ou outro desenvolvedor ocorre quando esse código é mesclado no repositório de códigos fonte.

Minha equipe configurou esse pipeline com etapas automatizadas que implantam nossas alterações com segurança na produção, assim não precisamos observar cada implantação. O pipeline executa as alterações mais recentes por meio de um conjunto de testes e verificações de segurança da implantação. Essas etapas automatizadas evitam que os defeitos que afetam o cliente cheguem à produção e limitam o impacto dos defeitos nos clientes se alcançarem a produção. Como desenvolvedora, tenho confiança de que o pipeline implantará minha alteração na produção com cautela e segurança, sem que eu precise observar ativamente.

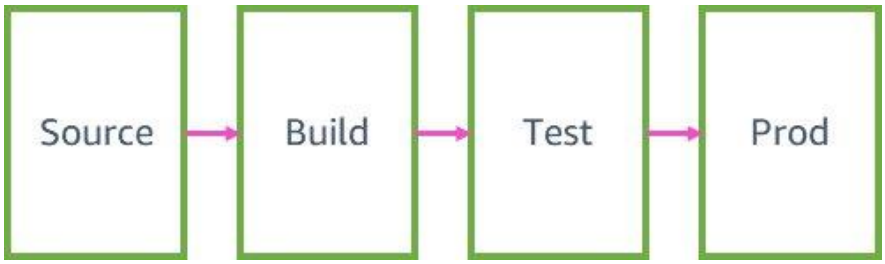
A jornada para a entrega contínua

A Amazon não começou praticando a entrega contínua, e os desenvolvedores aqui costumavam passar horas e dias gerenciando as implantações de seus códigos na produção. Adotamos a entrega contínua em toda a empresa como uma maneira de automatizar e padronizar a forma como implantamos software e para reduzir o tempo necessário para que as alterações chegassem à produção. As melhorias

em nosso processo de lançamento se acumularam gradativamente ao longo do tempo. Identificamos riscos à implantação e encontramos maneira de mitigá-los por meio da nova automação de segurança nos pipelines. Continuamos a repetir continuamente o processo de lançamento identificando novos riscos e novas maneiras de melhorar a segurança da implantação. Para saber mais sobre a jornada para a entrega contínua e como continuamos a melhorar, consulte o artigo da Builder's Library Acelerando com a entrega contínua.

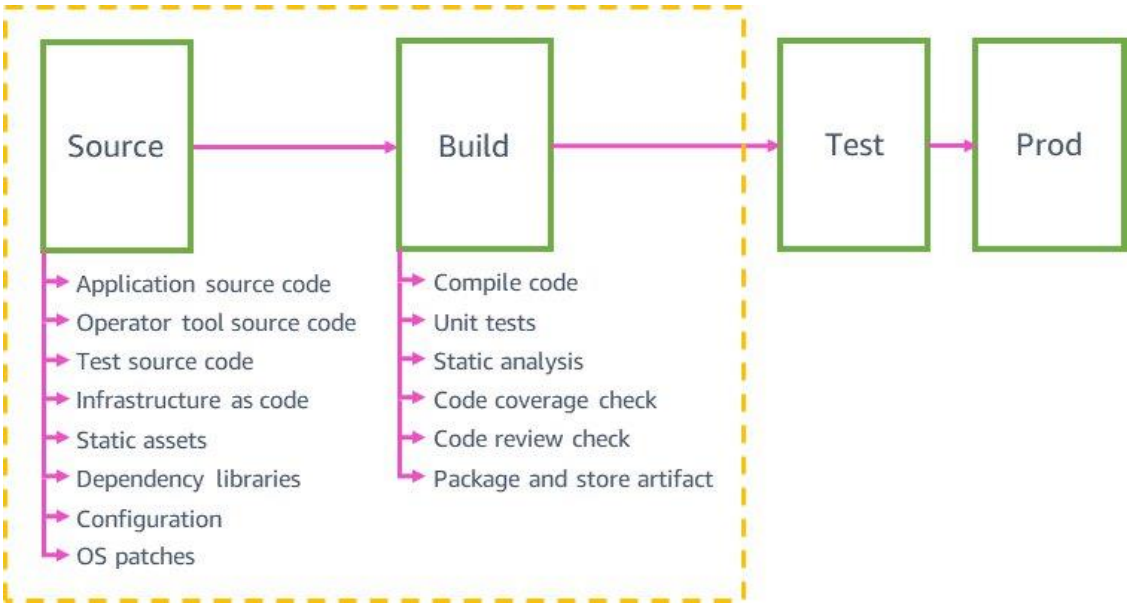
As quatro fases do pipeline

Neste artigo, explicamos as etapas pelas quais uma alteração de código passa em um pipeline na Amazon em seu caminho para a produção. Um pipeline típico de entrega contínua tem quatro fases importantes: origem, compilação, teste e produção (prod). Nos aprofundaremos no que acontece em cada uma desas fases do pipeline em um serviço típico da AWS e forneceremos um exemplo de como a equipe de um serviço típico da AWS pode configurar um de seus pipelines.



Fonte e compilação

O diagrama a seguir oferece uma visão geral das etapas das fontes e etapas de compilação que você pode encontrar em pipelines de equipes de serviços típicos da AWS.



Fontes do pipeline

Na Amazon, os pipelines validam automaticamente e implantam com segurança qualquer tipo de alteração de fonte na produção, não somente as alterações no código da aplicação. Eles podem validar e implantar alterações em fontes como ativos estáticos de sites, ferramentas, testes, infraestrutura, configuração e sistema operacional (SO) subjacente da aplicação. As versões de todas essas alterações são controladas em repositórios de códigos-fonte individuais. A dependências de códigos-fonte, como bibliotecas, linguagens de programação e parâmetros de programação como IDs de AMIs, recebem upgrades automaticamente para a versão mais recente pelo menos uma vez por semana.

Essas fontes são implantadas em pipelines individuais com os mesmos mecanismos de segurança (como reversão automática) que usamos para implantação do código da aplicação. Por exemplo, os valores de configuração de um serviço passível de ser alterado no tempo de execução (como aumentos de limites de taxas de APIs e sinalizações de recursos) são implantados automaticamente no pipeline de configuração dedicado. As alterações em fontes são revertidas automaticamente se causarem algum problema na produção em relação ao serviço (como falhas de interpretação de um arquivo de configuração).

Um microsserviço típico pode ter vários pipelines: do código da aplicação, de infraestrutura, de aplicação de patches no SO, de sinalizações de configurações/recursos e de ferramentas do operador. Ter vários pipelines para o mesmo microsserviço nos ajuda a implantar alterações na produção mais rapidamente. As alterações no código da aplicação que falham nos testes de integração e bloqueiam o pipeline da aplicação não afetam outros pipelines. Por exemplo, não impedem que as alterações no código da infraestrutura cheguem à produção no pipeline de infraestrutura. Todos os pipelines do mesmo microsserviço tendem a ter uma aparência muito semelhante. Por exemplo, um pipeline de sinalizações de recursos usa as mesmas técnicas de implantação segura que o pipeline do código da aplicação, já que uma alteração inválida na configuração de sinalizações de recursos pode ter o mesmo impacto na produção que uma alteração inválida no código da aplicação.

Revisão de código

Todas as alterações que vão para a produção começam com uma revisão de código e devem ser aprovadas por um membro da equipe antes de serem mescladas na ramificação da *linha principal* (nossa versão de “alimentador” ou “tronco”), que inicia o pipeline automaticamente. O pipeline impõe a exigência de que todas as confirmações na ramificação da linha principal devem ser revisadas e aprovadas por um membro da equipe de serviço desse pipeline. O pipeline impedirá que quaisquer confirmações não revisadas sejam implantadas.

Com pipelines totalmente automatizados, a revisão de código é a última revisão manual e aprovação que uma alteração de código recebe de um engenheiro antes de ser implantada na produção, portanto essa é uma etapa crítica. Os revisores de código avaliam a correção do código e também avaliam se a alteração pode ser implantada com segurança na produção. Eles avaliam se o código passou por testes suficientes (testes de unidade, testes de integração e testes canário), se está suficientemente instrumentado para monitoramento da implantação e se pode ser revertido com segurança. Algumas equipes usam uma lista de verificação personalizada, como mostra o exemplo a seguir, que é adicionada automaticamente a cada revisão de código da equipe para verificar explicitamente se há preocupações com a segurança na implantação.

Exemplo de lista de verificação de revisão de código

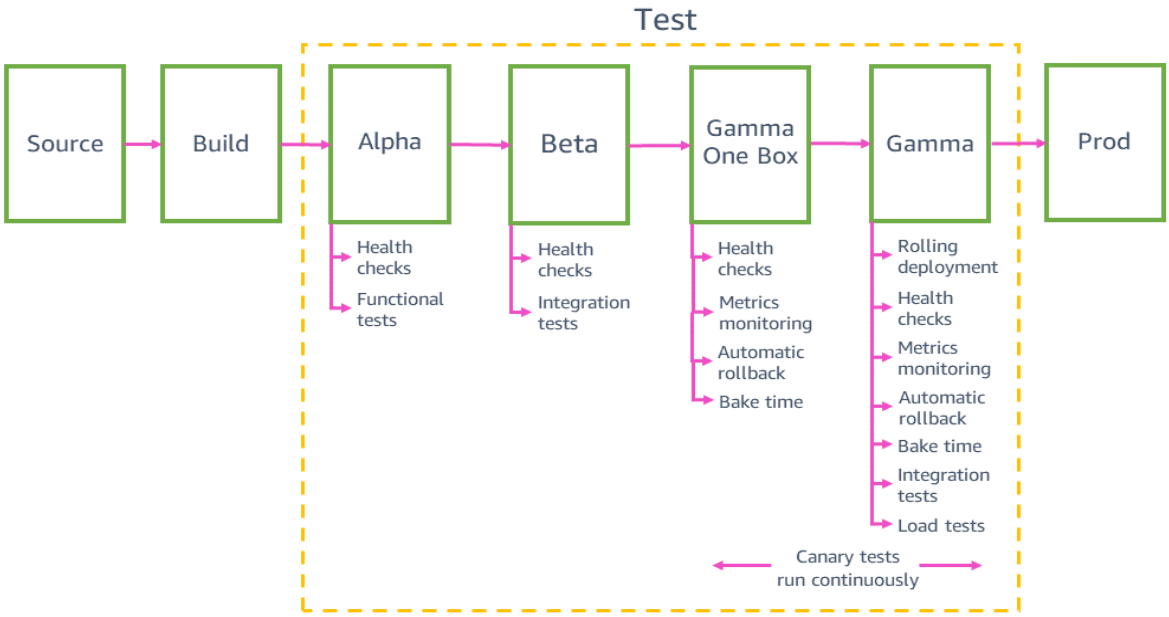
```
## Testing
[ ] Did you write new unit tests for this change?
[ ] Did you write new integration tests for this change?

Include the test commands you ran locally to test this change:
```
mvn test && mvn verify
```

## Monitoring
[ ] Will this change be covered by our existing monitoring?
 (no new canaries/metrics/dashboards/alarms are required)
[ ] Will this change have no (or positive) effect on resources and/or
limits?
 (including CPU, memory, AWS resources, calls to other services)
[ ] Can this change be deployed to Prod without triggering any alarms?

## Rollout
[ ] Can this change be merged immediately into the pipeline upon approval?
[ ] Are all dependent changes already deployed to Prod?
[ ] Can this change be rolled back without any issues after deployment to
Prod?
```

Compilação e testes de unidade

No estágio de compilação, o código é compilado e passa pelo teste de unidade. As ferramentas e a lógica da compilação podem variar de linguagem para linguagem e de equipe para equipe. Por exemplo, as equipes podem escolher estruturas de trabalho de teste de unidade, linters e ferramentas de análise estatística que funcionam melhor para elas. Além disso, as equipes podem escolher a configuração dessas ferramentas, como a cobertura mínima aceitável do código em sua estrutura de trabalho de teste de unidade. As ferramentas e os tipos de testes que serão executados também variam, dependendo do tipo de código implantado pelo pipeline. Por exemplo, os testes de unidade são usados para o código da aplicação e os linters são usados para modelos de infraestrutura como código. Todas as compilações são executadas sem acesso à rede para isolá-las e encorajar a reprodutibilidade delas. Normalmente, os testes de unidade simulam todas as chamadas à API deles para as dependências, como em outros serviços da AWS. As interações com dependências não simuladas “ativas” são testadas posteriormente no pipeline com os testes de integração. Comparados aos testes de integração, os testes de unidade com dependências simuladas são capazes de exercitar casos de borda, como erros inesperados retornados de chamadas à API e garantir o tratamento fluido de erros no código. Quando a compilação é concluída, o código compilado é empacotado e assinado.

Testar implantações em ambientes pré-produção

Antes de implantar na produção, o pipeline implanta e valida as alterações em vários ambientes pré-produção, por exemplo, alfa, beta e gama. O alfa e o beta validam se o código mais recente funciona como esperado executando testes funcionais de API e testes de integração de uma ponta a outra. O gama valida se o código é funcional e se pode ser implantado com segurança na produção. O gama é o ambiente mais parecido com o de produção possível, incluindo a mesma configuração de implantação, o mesmo monitoramento e alarmes, e os mesmos testes canário contínuos que os da produção. O gama também é implantado em várias regiões da AWS para capturar qualquer impacto em potencial das diferenças regionais.



Testes de integração

Os testes de integração nos ajudam a usar um serviço automaticamente, exatamente como os clientes fazem como parte do pipeline. Esses testes exercitam a pilha completa de uma ponta a outra chamando as APIs reais em execução na infraestrutura real em cada estágio pré-produção para todos os cenários significativos de clientes. A meta dos testes de integração é capturar qualquer comportamento inesperado ou incorreto do serviço antes da implantação na produção.

Embora os testes de unidade sejam executados em dependências simuladas, os testes de integração são executados em um sistema pré-produção que chama dependências reais, validando as hipóteses das simulações sobre como essas dependências se comportam. Os testes de integração validam o comportamento de APIs individuais entre diferentes entradas. Além disso, eles validam fluxos de trabalho completos que unem várias APIs, como a criação de um novo recurso, a descrição do novo recurso até que ele esteja pronto e, por fim, o uso do recurso.

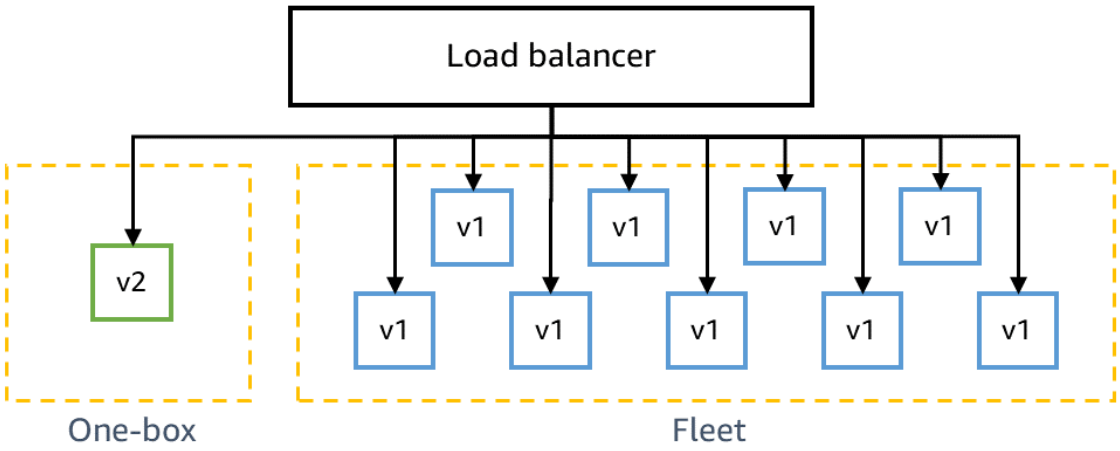
Os testes de integração executam casos de teste positivos e negativos, por exemplo, fornecendo uma entrada inválida a uma API e verificando se um erro de “entrada inválida” é retornado conforme

esperado. Alguns pipelines executam um teste difuso para gerar o máximo possível de entradas de API e validar se elas não causam alguma falha interna no serviço. Alguns pipelines também executam um breve teste de carga em um estágio pré-produção para garantir que as alterações mais recentes não causem nenhuma latência ou regressões de taxa de transferência em níveis de carga reais.

Compatibilidade retroativa e testes one-box

Antes de implantar na produção, precisamos garantir que o código mais recente seja retrocompatível e possa ser implantado com segurança junto ao código atual. Por exemplo, precisamos detectar se o código mais recente grava dados em um formato que o código atual não consegue analisar. O estágio *one-box* no ambiente gama implanta o código mais recente na menor unidade de implantação, como em uma única máquina virtual ou contêiner, ou em uma pequena porcentagem de invocações de funções do AWS Lambda. Essa implantação one-box deixa o restante do ambiente gama implantado com o código atual por algum tempo, como 30 minutos ou 1 hora. O tráfego não tem de ser especificamente dirigido para a one-box. Ele pode ser adicionado ao mesmo load balancer ou consultar a mesma fila que o restante do ambiente gama. Por exemplo, em um ambiente gama de dez contêineres atrás de um load balancer, a one-box recebe 10% do tráfego gama gerado por testes canário contínuos. A implantação one-box monitora as taxas de sucesso de testes canário e as métricas de serviço para detectar qualquer impacto da implantação ou de ter uma frota "mista" implantada lado a lado.

O diagrama a seguir mostra o estado de um ambiente gama após o novo código ter sido implantado no estágio one-box, mas ainda não implantado no restante da frota gama:



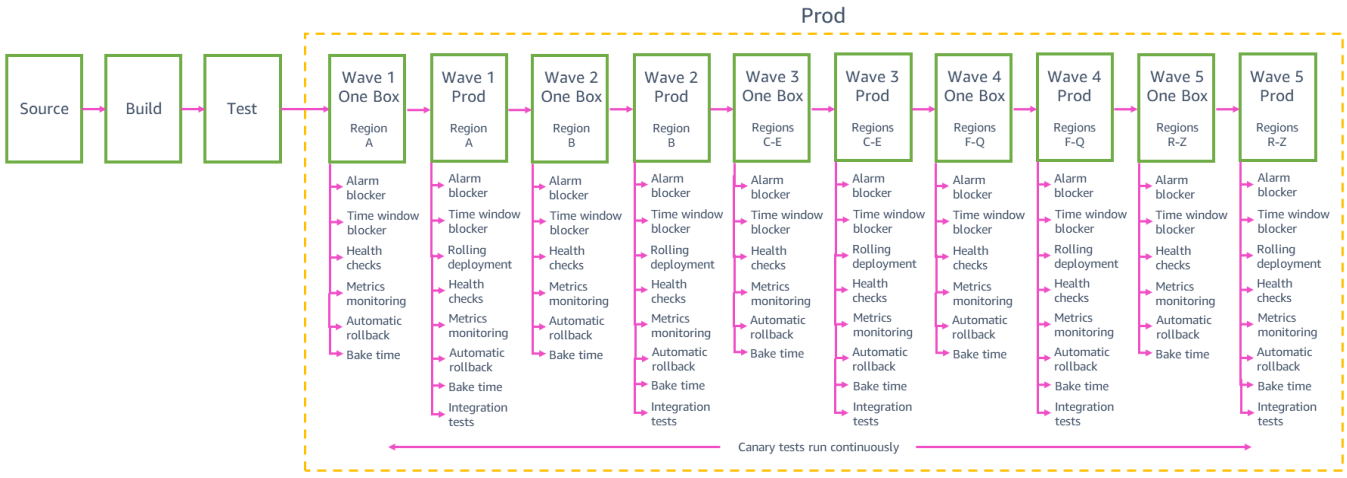
Também precisamos garantir que o código mais recente seja compatível com as versões anteriores de nossas dependências, por exemplo, se uma alteração precisar ser feita nos microsserviços em uma ordem específica. Os microsserviços em ambientes de pré-produção normalmente chamam o endpoint de produção de quaisquer serviços pertencentes a outra equipe, como Amazon Simple Storage Service (S3) ou Amazon DynamoDB, mas eles chamam o endpoint de pré-produção de outros microsserviços da equipe de serviço no mesmo estágio. Por exemplo, o microsserviço A de uma equipe em gama chama o microsserviço B da mesma equipe em gama, mas chama o endpoint de produção do Amazon S3.

Alguns pipelines também executam testes de integração novamente em um estágio separado de compatibilidade com versões anteriores que chamamos de *zeta*, que é um ambiente separado onde cada microsserviço chama apenas endpoints de produção, testando se as alterações que vão para a produção são compatíveis com o código atualmente implantado na produção em vários microsserviços. Por exemplo, o microsserviço A em zeta chama o endpoint de produção do microsserviço B e o endpoint de produção para o Amazon S3.

Para obter uma descrição das estratégias para criar e implantar alterações retrocompatíveis, consulte o artigo da Builders 'Library Garantindo a segurança de reversão durante as implantações.

Implantações na produção

Nosso objetivo nº 1 para implantações de produção na AWS é evitar o impacto negativo em várias regiões ao mesmo tempo e em várias zonas de disponibilidade na mesma região. Limitar o escopo de cada implantação individual limita o impacto potencial nos clientes devido a implantações de produção com falha e evita um impacto em várias zonas de disponibilidade ou em várias regiões. Para limitar o escopo das implantações automáticas, dividimos a fase de produção do pipeline em vários estágios e muitas implantações em regiões individuais. As equipes dividem as implantações regionais em implantações de escopo ainda menor, implantando em zonas de disponibilidade individuais ou em fragmentos internos individuais de seu serviço (chamados de *células*) em seu pipeline, para limitar ainda mais o escopo do impacto potencial de uma implantação de produção com falha.



Implantações escalonadas

Cada equipe precisa equilibrar a segurança de implantações com pequeno escopo e a velocidade com que podemos entregar alterações aos clientes em todas as regiões. A implantação de alterações em 24 regiões ou 76 zonas de disponibilidade por meio do pipeline, uma de cada vez, tem o menor risco de causar amplo impacto, mas pode levar semanas para que o pipeline entregue uma alteração aos clientes globalmente. Descobrimos que agrupar implantações em "ondas" de tamanho crescente, como visto no exemplo anterior de pipeline de produção, nos ajuda a alcançar um bom equilíbrio entre risco de implantação e velocidade. O estágio de cada onda no pipeline orquestra implantações em um

grupo de regiões, com alterações sendo promovidas de onda para onda. Novas alterações podem entrar na fase de produção do pipeline a qualquer momento. Depois que um conjunto de alterações é promovido da primeira etapa para a segunda etapa na onda 1, o próximo conjunto de alterações provenientes do ambiente gama é promovido para a primeira etapa da onda 1, então não terminamos com grandes pacotes de alterações esperando para serem implantados na produção.

As duas primeiras ondas no pipeline geram mais confiança na alteração: a primeira onda é implantada em uma região com um baixo número de solicitações para limitar o possível impacto resultante da primeira implantação de produção da nova alteração. A onda é implantada em apenas uma zona de disponibilidade (ou célula) por vez dentro dessa região para implantar cautelosamente a alteração em toda a região. A segunda onda é então implantada em uma zona de disponibilidade (ou célula) por vez em uma Região com um grande número de solicitações onde é altamente provável que os clientes exercitem todos os novos caminhos de código e onde obtenhamos uma boa validação das alterações.

Depois que tivermos maior confiança na segurança da alteração após as implantações das ondas do pipeline inicial, podemos implantar em mais e mais regiões em paralelo na mesma onda. Por exemplo, o exemplo de pipeline de produção anterior é implantado em três regiões na onda 3, depois em até 12 regiões na onda 4 e, em seguida, nas regiões restantes na onda 5. O número exato e a escolha de regiões em cada uma dessas ondas e o número de ondas no pipeline de uma equipe de serviço dependem dos padrões de uso e da escala do serviço individual. As ondas posteriores no pipeline ainda nos ajudam a atingir nosso objetivo de evitar o impacto negativo em várias zonas de disponibilidade na mesma região. Quando uma onda é implantada em várias regiões em paralelo, ela segue o mesmo comportamento de implantação cauteloso para cada região que foi usada nas ondas iniciais. Cada etapa da onda só é implantada em uma única zona de disponibilidade ou célula de cada região da onda.

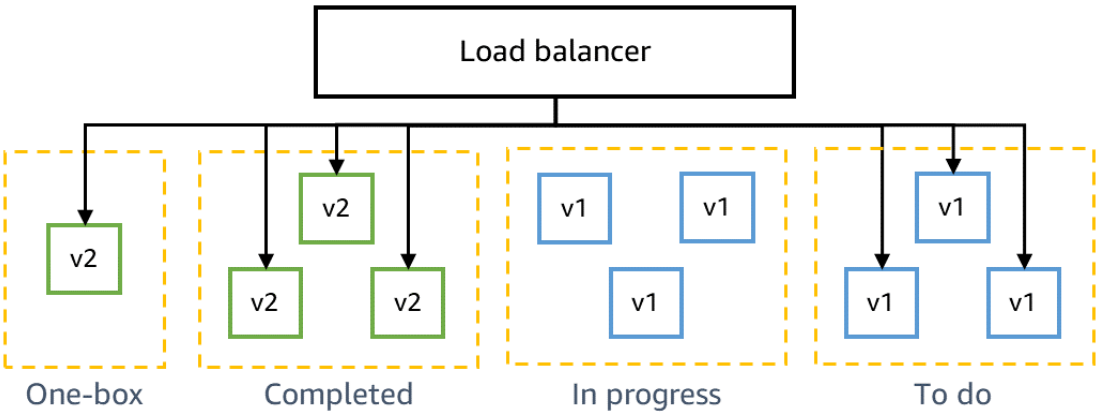
Implantações one-box e contínuas

As implantações em cada onda de produção começam com um estágio one-box. Como no estágio one-box do ambiente gama, cada estágio one-box de produção implanta o código mais recente em uma *caixa* (uma única máquina virtual, um único contêiner ou uma pequena porcentagem de invocações de funções do Lambda) em cada uma das regiões ou zonas de disponibilidade da onda. A implantação one-box de produção minimiza o impacto potencial das alterações na onda, inicialmente limitando as solicitações atendidas pelo novo código nessa onda. Normalmente, a one-box atende a no máximo dez por cento das solicitações gerais para a região ou zona de disponibilidade. Se a alteração causar um impacto negativo na one-box, o pipeline reverte automaticamente a alteração e não a promove para o restante dos estágios de produção.

Após o estágio one-box, a maioria das equipes usa implantações contínuas para implantar na frota de produção principal da onda. Uma implantação contínua garante que o serviço tenha capacidade suficiente para atender à carga de produção durante toda a implantação. Ela controla a taxa na qual o novo código é *colocado em serviço* (ou seja, quando começa a atender ao tráfego de produção) para limitar o impacto das alterações. Em uma implantação típica em uma região, no máximo 33% das caixas do serviço naquela região (contêineres, invocações do Lambda ou software em execução em máquinas virtuais) são substituídas pelo novo código.

Durante uma implantação, o sistema de implantação escolhe primeiro um lote inicial de até 33% das caixas para substituir pelo novo código. Durante a substituição, pelo menos 66% da capacidade geral está íntegra e atendendo às solicitações. Todos os serviços são dimensionados para resistir à perda de uma zona de disponibilidade na região, portanto, sabemos que o serviço ainda pode atender à carga de produção com essa capacidade. Depois que o sistema de implantação determina que uma caixa no lote inicial de caixas está passando nas verificações de integridade, uma caixa da frota restante pode ser substituída pelo novo código e assim por diante. Enquanto isso, ainda mantemos um mínimo de 66% da capacidade para atender às solicitações o tempo todo. Para limitar ainda mais o impacto das alterações, os pipelines de algumas equipes implantam apenas 5% de suas caixas de cada vez. No entanto, eles fazem *reversões* rápidas, em que o sistema substitui 33% das caixas de uma vez pelo código anterior para acelerar a reversão.

O diagrama a seguir mostra o estado de um ambiente de produção no meio de uma implantação contínua. O novo código foi implantado no estágio one-box e no primeiro lote da frota de produção principal. Outro lote foi removido do load balancer e está sendo encerrado para substituição.



Monitoramento de métricas e reversão automática

As implantações automatizadas no pipeline normalmente não têm um desenvolvedor que monitora ativamente cada implantação na produção, verifica as métricas e reverte manualmente se houver problemas. Essas implantações não exigem nenhuma intervenção manual. O sistema de implantação monitora ativamente um alarme para determinar se ele precisa reverter automaticamente uma implantação. Uma reversão mudará o ambiente de volta para a imagem do contêiner, pacote de implantação de funções do AWS Lambda ou pacote de implantação interno que foi implantado anteriormente. Nossos pacotes de implantação internos são semelhantes às imagens de contêiner, porque os pacotes são imutáveis e usam uma soma de verificação para verificar sua integridade.

Cada microsserviço em cada região normalmente tem um alarme de alta gravidade que dispara nos limites das métricas que afetam os clientes do serviço (como taxas de falha e alta latência) e nas métricas de integridade do sistema (como utilização de CPU), conforme ilustrado no exemplo a seguir. Este alarme de alta gravidade é usado para enviar uma mensagem ao engenheiro de plantão e reverter automaticamente o serviço se uma implantação estiver em andamento. Muitas vezes,

a reversão já está em andamento no momento em que o engenheiro de plantão foi avisado e começa a se envolver.

Exemplo de alarme de microsserviço de alta gravidade

```
ALARM("FrontEndApiService_High_Fault_Rate") OR
ALARM("FrontEndApiService_High_P50_Latency") OR
ALARM("FrontEndApiService_High_P90_Latency") OR
ALARM("FrontEndApiService_High_P99_Latency") OR
ALARM("FrontEndApiService_High_Cpu_Usage") OR
ALARM("FrontEndApiService_High_Memory_Usage") OR
ALARM("FrontEndApiService_High_Disk_Usage") OR
ALARM("FrontEndApiService_High_Errors_In_Logs") OR
ALARM("FrontEndApiService_High_Failing_Health_Checks")
```

As alterações introduzidas por uma implantação podem ter um impacto nos microsserviços em etapas anteriores e posteriores do processo, portanto, o sistema de implantação precisa monitorar o alarme de alta gravidade para o microsserviço em implantação *e* monitorar os alarmes de alta gravidade para os outros microsserviços da equipe para determinar quando reverter. As alterações implantadas também podem afetar as métricas do teste canário contínuo, portanto, o sistema de implantação também precisa monitorar a falha nos testes canário. Para reverter automaticamente em todas essas áreas possíveis de impacto, as equipes criam alarmes agregados de alta gravidade para o sistema de implantação monitorar. Os alarmes agregados de alta gravidade acumulam o estado de todos os alarmes de alta gravidade do microsserviço individual da equipe e o estado dos alarmes canário em um único estado agregado, como no exemplo a seguir. Se algum dos alarmes de alta gravidade para os microsserviços da equipe entrar em estado de alarme, todas as implantações em andamento da equipe em todos os seus microsserviços naquela região serão revertidas automaticamente.

Exemplo de alarme de reversão agregado de alta gravidade

```
ALARM("FrontEndApiService_High_Severity") OR
ALARM("BackendApiService_High_Severity") OR
ALARM("BackendWorkflows_High_Severity") OR
ALARM("Canaries_High_Severity")
```

Um estágio one-box atende a uma pequena porcentagem do tráfego geral, portanto, os problemas introduzidos por uma implantação one-box podem não acionar o alarme de reversão de alta gravidade do serviço. Para capturar e reverter as alterações que causam problemas no estágio one-box antes de atingirem o restante dos estágios de produção, os estágios de uma caixa, adicionalmente, retrocedem nas métricas que têm como escopo apenas o one-box. Por exemplo, eles revertem a taxa de falha das solicitações que foram atendidas especificamente pelo one-box, o que representa uma pequena porcentagem do número geral de solicitações.

Exemplo de alarme de reversão de one-box

```
ALARM("High_Severity_Aggregate_Rollback_Alarm") OR
ALARM("FrontEndApiService_OneBox_High_Fault_Rate") OR
ALARM("FrontEndApiService_OneBox_High_P50_Latency") OR
ALARM("FrontEndApiService_OneBox_High_P90_Latency") OR
ALARM("FrontEndApiService_OneBox_High_P99_Latency") OR
ALARM("FrontEndApiService_OneBox_High_Cpu_Usage") OR
ALARM("FrontEndApiService_OneBox_High_Memory_Usage") OR
ALARM("FrontEndApiService_OneBox_High_Disk_Usage") OR
ALARM("FrontEndApiService_OneBox_High_Errors_In_Logs") OR
ALARM("FrontEndApiService_OneBox_Failing_Health_Checks")
```

Além de reverter os alarmes definidos pela equipe de serviço, nosso sistema de implantação também pode detectar e reverter automaticamente as anomalias em métricas comuns emitidas por nossa estrutura de trabalho interna do serviço Web. A maioria dos nossos microsserviços emite métricas, como contagem de solicitações, latência de solicitações e contagem de falhas em um formato padrão. Usando essas métricas padrão, o sistema de implantação pode reverter automaticamente se houver anomalias nas métricas durante uma implantação. Exemplos disso são se a contagem de solicitações cair repentinamente para zero ou se a latência ou o número de falhas se tornar muito maior do que o normal.

Tempo de incorporação

Às vezes, um impacto negativo causado por uma implantação não é imediatamente aparente. É uma *combustão lenta*. Ou seja, ele não aparece imediatamente durante a implantação, especialmente se o serviço estiver sob carga baixa no momento. A promoção da alteração para o próximo estágio do pipeline imediatamente após a conclusão da implantação pode acabar tendo um impacto em várias regiões no momento em que o impacto surge na primeira região. Antes de promover uma alteração para o próximo estágio de produção, cada estágio de produção no pipeline tem *tempo de incorporação*, que é quando o pipeline continua a monitorar o alarme agregado de alta gravidade da equipe para qualquer impacto de queima lenta após a conclusão de uma implantação e antes de passar para o próximo estágio.

Para calcular o tempo que gastamos preparando uma implantação, precisamos equilibrar o risco de causar um impacto mais amplo se promovermos alterações em várias regiões muito rapidamente em relação à velocidade na qual podemos entregar alterações aos clientes globalmente. Descobrimos que uma boa maneira de equilibrar esses riscos é que as ondas anteriores no pipeline tenham um tempo de incorporação mais longo enquanto construímos a confiança na segurança da alteração e, em seguida, as ondas posteriores tenham um tempo de incorporação mais abreviado. Nosso objetivo é minimizar o risco de um impacto que afete várias regiões. Como a maioria das implantações não é monitorada ativamente por um membro da equipe, os tempos de incorporação padrão do pipeline são conservadores e implantarão uma alteração em todas as regiões em cerca de quatro ou cinco dias úteis. Serviços maiores ou altamente críticos têm tempos de incorporação ainda mais conservadores e tempos para que seus pipelines implantem uma alteração globalmente.

Um pipeline típico espera pelo menos uma hora após cada estágio one-box, pelo menos 12 horas após a primeira onda regional e pelo menos duas a quatro horas após cada uma das demais ondas regionais, com tempo de incorporação adicional para regiões individuais, zonas de disponibilidade e células dentro de cada onda. O tempo de incorporação inclui requisitos para esperar por um número específico de pontos de dados nas métricas da equipe (por exemplo, "aguarde pelo menos 100 solicitações para a API Create") para garantir que ocorreram solicitações suficientes para tornar provável que o novo código foi totalmente exercido. Durante todo o tempo de incorporação, a implantação é revertida automaticamente se o alarme agregado de alta gravidade da equipe entrar for acionado.

Embora seja extremamente raro, em alguns casos, uma alteração urgente (como uma correção de segurança ou uma atenuação para um evento de grande escala que afete a disponibilidade do serviço) pode precisar ser entregue aos clientes mais rapidamente do que o tempo que o pipeline normalmente leva para incorporar as alterações e implantá-las. Nesses casos, podemos reduzir o tempo de incorporação do pipeline para acelerar a implantação, mas exigimos um alto nível de análise da alteração para fazer isso. Para esses casos, exigimos o escrutínio dos engenheiros-chefe da organização. A equipe deve revisar a alteração de código, bem como sua urgência e risco de impacto, com desenvolvedores muito experientes que são especialistas em segurança operacional. A alteração ainda passa pelas mesmas etapas de costume no pipeline, mas é promovida para o próximo estágio mais rapidamente. Gerenciamos o risco de uma implantação mais rápida limitando as alterações em andamento no pipeline durante esse tempo para permitir apenas as mínimas alterações de código necessárias para resolver o problema atual e observando ativamente as implantações.

Bloqueadores de alarmes e janelas de tempo

O pipeline evita implantações automáticas na produção quando há um risco maior de causar um impacto negativo. O pipeline usa um conjunto de “bloqueadores” que avaliam o risco da implantação. Por exemplo, implantar automaticamente uma nova alteração na produção quando um problema está em andamento no ambiente pode piorar ou prolongar o impacto. Antes de iniciar uma nova implantação em qualquer estágio de produção, o pipeline verifica o alarme agregado de alta gravidade da equipe para determinar se há algum problema ativo. Se o alarme estiver atualmente no estado de alarme, o pipeline impede que a alteração avance. Os pipelines também podem verificar alarmes em toda a organização, como um alarme de evento em grande escala que indica se há um amplo impacto nos sistemas de outra equipe e evita o início de uma nova implantação que poderia aumentar o impacto geral. Esses bloqueadores de implantação podem ser substituídos por desenvolvedores quando uma alteração precisa ser implantada na produção para a recuperação de um problema de alta gravidade.

O pipeline também é configurado com um conjunto de janelas de tempo que definem quando uma implantação pode começar. Quando configuramos janelas de tempo, precisamos equilibrar duas causas de risco de implantação. Por um lado, janelas de tempo muito pequenas podem causar o acúmulo de alterações no pipeline enquanto a janela de tempo estiver fechada, aumentando a probabilidade de que qualquer uma dessas alterações na próxima implantação venha a causar impacto quando a janela de tempo abrir. Por outro lado, janelas de tempo muito grandes, que vão além do horário comercial normal, aumentam o risco de prolongar o impacto de uma implantação com falha. Durante o expediente, leva mais tempo para envolver o engenheiro de plantão do que durante o dia, quando o engenheiro de plantão e outros membros da equipe estão trabalhando.

Durante o horário comercial normal, a equipe pode ser envolvida mais rapidamente após uma implantação com falha, caso sejam necessárias quaisquer etapas de recuperação manual.

A maioria das implantações não é monitorada ativamente por um membro da equipe, portanto, otimizamos o tempo das implantações para minimizar o tempo necessário para envolver um engenheiro de plantão caso haja uma ação manual necessária para a recuperação após uma reversão automática. Os engenheiros de plantão geralmente demoram mais para se envolver à noite, nos feriados e nos finais de semana, portanto, esses horários são excluídos das janelas de tempo. Dependendo dos padrões de uso do serviço, alguns problemas podem não aparecer por horas após a implantação, portanto, muitas equipes também excluem as implantações de sexta-feira e final de tarde de suas janelas de tempo para reduzir o risco de precisar envolver o engenheiro de plantão à noite ou durante no fim de semana após uma implantação. Descobrimos que esse conjunto de janelas de tempo permite uma recuperação rápida mesmo quando a ação manual é necessária, garante menos envolvimento com os engenheiros de plantão fora do expediente normal e garante que um pequeno número de alterações seja agrupado enquanto as janelas de tempo são fechadas .

Pipelines como código

A equipe típica de serviços da AWS tem muitos pipelines para implantar os vários microsserviços e tipos de fonte da equipe (código da aplicação, código da infraestrutura, patches do sistema operacional, etc.). Cada pipeline tem muitos estágios de implantação para um número cada vez maior de regiões e zonas de disponibilidade. Isso se traduz em muita configuração para a equipe gerenciar no sistema do pipeline, no sistema de implantação e no sistema de alarme, e muito esforço para se manter atualizada sobre as melhores práticas mais recentes e as novas regiões e zonas de disponibilidade. Nos últimos anos, adotamos a prática de “pipelines como código” como uma forma de configurar pipelines seguros e atualizados de forma mais fácil e consistente modelando essa configuração em código. Nossa ferramenta interna de pipelines como código extrai de uma lista centralizada de regiões e zonas de disponibilidade para adicionar facilmente novas regiões e zonas de disponibilidade aos pipelines em toda a AWS. A ferramenta também permite que as equipes modelem pipelines usando herança, definindo a configuração comum entre os pipelines de uma equipe em uma classe pai (como quais regiões vão em cada onda e quanto tempo de incorporação deve ser para cada onda) e definindo todas as configurações de pipeline de microsserviços como uma subclasse que herda todas as configurações comuns.

Conclusão

Na Amazon, criamos nossas práticas de implantação automatizadas ao longo do tempo, com base no que nos ajuda a equilibrar a segurança da implantação com a velocidade em que ela é realizada. Ao mesmo tempo, queremos minimizar o tempo que os desenvolvedores passam se preocupando com as implantações. Criar a segurança de implantação automatizada no processo de lançamento usando testes extensivos de pré-produção, reversões automáticas e implantações de produção escalonadas nos permite minimizar o impacto potencial na produção causado pelas implantações. Isso significa que os desenvolvedores não precisam observar ativamente as implantações na produção.

Com pipelines totalmente automatizados, os desenvolvedores usam revisões de código para verificar seu código e também para aprovar se a alteração está pronta para entrar em produção. Depois que a alteração é incorporada ao repositório de códigos-fonte, o desenvolvedor pode passar para

a próxima tarefa e esquecer a implantação, confiando no pipeline para colocar sua alteração em produção com segurança e cautela. O pipeline automatizado se encarrega de implantar continuamente na produção várias vezes ao dia, enquanto equilibra segurança e velocidade. Ao modelar nossa prática de entrega contínua em código, é mais fácil do que nunca para as equipes de serviço da AWS configurarem seus pipelines para implantar suas alterações de código de forma automática e segura.