Implantação Contínua DevOps

A implantação contínua e a automatização das etapas de build e testes, alteração de código, etapas do pipeline e automação do ambiente de produção sem intervenções manuais.

Brunelli Gabriel Cupello

1. Itens iniciais

Propósito

O conhecimento da implantação contínua DevOps, é essencial para otimizar a qualidade e a rapidez na produção de softwares, trazendo maiores lucros para sua empresa e tornando o cliente mais satisfeito. Além disso, é um atributo mais atrativo para o mercado de trabalho, uma vez que permite ao seu cliente fazer upgrades em aplicativos e/ou programas de forma mais rápida, barata, segura e eficaz.

Objetivos

- Identificar o pipeline de implantação de software.
- Identificar a aplicação do commit de uma nova versão de software.
- Reconhecer os testes de aplicação.
- Aplicar corretamente a implantação e a entrega de software.

Introdução

A implementação do DevOps envolve a adoção de medidas para a integração contínua com testagem bem realizada e etapas de entrega contínua, e, caso não haja detecção de problemas, o serviço entra em funcionamento.

A implementação contínua é uma extensão da integração contínua, cujo foco é garantir que o software seja sempre o mais atualizado para seus clientes, enviando automaticamente as alterações de código de forma rápida e confiável. Um de seus diferenciais é um processo em que não ocorre nenhuma intervenção manual, acelerando a disponibilidade do software para o usuário final, diferentemente da implementação tradicional.

Toda a infraestrutura de implantação contínua está sempre monitorando o software, se em algum momento acontecer uma falha no comportamento geral do software, um mecanismo automático reverte as alterações, restaurando-o para a versão original, graças aos testes automatizados que são fundamentais na hora da implantação da fundação da integração contínua.

Acompanhe no vídeo o processo de implantação contínua em DevOp.

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

O esqueleto da pipeline de implantação

Confira os passos do processo de implantação contínua e as boas práticas de implantação do pipeline e dos testes.

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Não há dúvidas de que a integração contínua contribui muito para o aumento da qualidade e produtividade dos projetos que a adotaram, uma vez que ela faz as equipes trabalharem juntas para criarem sistemas complexos com alta confiabilidade. Mas a integração contínua é focada na área de desenvolvimento (ela busca ter certeza de que o código condiz com os parâmetros estabelecidos e tenha sucesso em um número de testes), o que gera um produto que ainda irá passar por um teste manual seguindo para o lançamento.

Muito do tempo da criação de um software é gasto entre testes e desenvolvimento, pela natureza manual desse campo. Há casos em que pode acontecer de testers ficarem esperando builds “boas” do software, equipes de desenvolvimento recebendo feedback semanas depois de já começarem uma nova funcionalidade ou até descobrirem problemas apenas no fim do ciclo do desenvolvimento.

Todo o processo de criar o software, desenvolver, progredir as builds por meio de múltiplos estágios de testes até a implantação, necessita da colaboração de muitos indivíduos e múltiplas equipes.

Vejamos:

Pipeline de implantação

A pipeline de implantação modela todo esse processo, e sua incorporação em uma integração contínua e ferramenta de gerenciamento de lançamento é o que permite que você veja e controle o progresso de cada mudança conforme ela progride da versão de controle por meio de vários testes até ser lançada para o usuário.

Início da implantação

O processo de implantação da pipeline começa com os desenvolvedores dando commit nas mudanças no sistema de controle de versão. Nesse momento, o sistema de gerenciamento da integração contínua irá criar uma nova instância da nossa pipeline. O primeiro estágio da pipeline (commit) compila o código, roda testes de unidade, realiza análise do código e cria instaladores.

Armazenamento de artefatos

Se todos os testes forem um sucesso e o código estiver apto a avançar, então passamos o código executável para binário e o guardamos em repositórios de artefato. A maioria dos servidores de implantação contínua fornece facilities para guardar artefatos como esse para serem facilmente acessados tanto pelos usuários, quanto para os próximos estágios da sua pipeline.

Outro detalhe é que a maioria dos servidores de implantação contínua permite que atividades, como preparo da data-base de teste para uso em testes de aceitação, sejam executadas em paralelo em uma grid de build.

O segundo estágio será iniciado automaticamente após a conclusão (com sucesso) do primeiro, sendo esse responsável pelos testes de aceitação automatizados de longa duração. Nesse momento, o seu servidor de implantação contínua deve permitir que você realize esses testes em suítes que podem ser executadas em paralelo para aumentar a rapidez do feedback e velocidade dos testes.

A partir desse momento, não é recomendado que o próximo estágio seja iniciado automaticamente, pois, nesse caso, estaríamos falando de testes de produção, capacidade e teste de aceitação do usuário. O ideal é que os testers ou equipes de operação atuem manualmente com builds self-service, facilitando o processo com scripts automatizados que executam essa implantação.

Saiba mais

No software de entrega contínua, será possível visualizar os candidatos para lançamento junto com o status (se há algum comentário de check-in, sucesso nos estágios ou qualquer outro comentário), podendo então com o aperto de um botão implantar a build selecionada através de um script de implantação no ambiente relevante. Esse mesmo critério será utilizado em mais estágios da pipeline, porém, com um grupo de usuários restrito em cada ambiente responsável por atuar as builds self-service, como a equipe de operações ser responsável pela aprovação de builds sendo aplicadas no ambiente de produção.

Poder verificar e correlacionar um check-in em particular, versão da build e estágios da pipeline que já se passaram é crucial para um bom sucesso da implantação contínua, pois se você encontrar uma falha em um teste, como de performance, você pode checar imediatamente quais mudanças foram feitas que acabaram dando resultado negativo no teste.

Boas práticas no pipeline de implantação

Confira alguns aspectos relevantes sobre este assunto.

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Em relação à pipeline de implantação, Humble e Farley (2010) citam vários métodos que podem ser utilizados para garantir um workflow mais eficiente. Por exemplo, “binários” podem ser referidos como uma coleção de códigos executáveis, mas, caso não seja necessário compilar o código, esses “binários” podem ser representados por coleções de arquivos fonte, como assemblies.NET, Jars etc.

Vejamos três boas práticas!

Só faça seus binários uma vez (Only build your binaries once)

Antes de começarmos, é necessário lembrar que há dois princípios importantes da pipeline de implantação:

Manter...

Manter a pipeline eficiente para que a equipe receba feedback o mais rápido possível.

Criar...

Criar builds em fundações sólidas sempre.

Há casos em que sistemas de builds irão usar o código fonte na versão de controle como a fonte para muitos passos.

> Esse código será compilado muitas vezes em vários contextos (commit, teste de aceitação de tempo, capacidade etc).

Porém, toda vez que você compila o código, há a possibilidade de introduzir alguma diferença. Essa diferença pode acontecer em várias situações, como, talvez, a configuração do compilador mude o comportamento da aplicação ou você usou não intencionalmente uma versão de uma biblioteca externa.

Todo esse processo de recompilação quebra o primeiro fundamento da pipeline porque é um processo que precisa de tempo, principalmente quando se trata de sistemas grandes.

Essa diferença que pode ser introduzida também é responsável por não estar de acordo com o segundo fundamento: todos os binários que passaram pelo teste de aceitação devem ser exatamente os mesmos que chegam na produção. Na pipeline isso pode ser checado por hashes nos binários no momento em que eles são criados e verificados em todos os estágios que eles estão inalterados durante o processo em geral.

> É vital para o sistema que essa verificação exista, pois, caso os binários sejam recriados ao invés de reusados na hora da utilização, você passa a criar a possibilidade de uma mudança (tanto intencional ou por erro).

Algumas empresas possuem a cultura de que assembly e compilação só podem ocorrer em ambientes controlados onde apenas os funcionários seniors podem acessar.

Implante do mesmo jeito para todos os ambientes (Deploy the same way to every environment)

Todo ambiente é diferente, porém o script de implantação precisa ter a mesma metodologia para todos. Humble e Farley (2010) explicam que, mesmo que todos os ambientes sejam diferentes, não significa que o script de implantação seja diferente para cada um deles.

Pelo contrário, ao utilizar o método igual para todos (uma estação de trabalho de um analista, um ambiente de teste etc), você passa a garantir que o processo de implantação é testado propriamente.

Ao agregar as informações de configuração nos arquivos de propriedade, você passa a linkar as configurações que são únicas para cada ambiente separado e serve como caminho a se seguir na hora de usar o mesmo script de implantação.

Só que esse script precisará ter uma variável de ambiente associada a ele (caso esteja em um ambiente de várias máquinas), pois será por meio dessa variável que a versão de controle, que irá checar os arquivos, vai poder selecionar o script correto ou, no caso de um ambiente de servidor local, pelo hostname.

Exemplo

Uma empresa na qual os ambientes de produção são gerenciados por uma equipe diferente da equipe responsável pelo ambiente de desenvolvimento e teste. Nesse caso, as duas equipes terão que trabalhar para poder garantir a sincronia e que o processo de implantação automática será bem-sucedido em todos os ambientes, incluindo os de desenvolvimento.

Usar o mesmo script garante que você conseguirá testar o processo de implantação várias vezes enquanto implanta para todos os ambientes, aumentando a confiabilidade, diminuindo o risco de lançamento do software e de prevenção para que o software funcione em algumas máquinas e outras não.

Ao utilizar o mesmo script de implantação para todos os ambientes, você passa a ter muito mais controle em relação a algum problema na hora de um mau funcionamento, pois só irá ter três caminhos a se checar:

- Alguma configuração no arquivo específico do ambiente em que está se trabalhando.
- Um problema na infraestrutura/serviço que a sua aplicação precisa.
- A configuração do seu ambiente.

Teste de fumaça das implantações e implante de uma cópia da produção

Você precisa de um script automatizado que faça um teste de fumaça toda vez que você implantar uma aplicação para garantir que ele está funcionando normalmente, podendo ser algo básico, como abrir a aplicação e a tela principal está com o conteúdo esperado.

Esses tipos de testes de fumaça fazem com que a confiabilidade de todo o seu sistema aumente consideravelmente, já que você consegue ter feedback em tempo real se a sua aplicação está funcionando ou não, fornecendo diagnósticos como dizer se alguma coisa que ela depende está off-line.

> “
>
> Ao efetuar os testes, é sempre bom ter um ambiente o mais próximo possível da produção. Não é raro encontrar equipes tendo problemas na hora de lançar um aplicativo no qual o ambiente de produção é diferente do ambiente de teste ou de desenvolvimento.
>
> —
>
> (HUMBLE; FARLEY, 2010)

Utilizar um ambiente parecido com o da produção diminui a taxa de possíveis erros na hora de lançar o aplicativo. Mas é necessário um bom gerenciamento na hora de criar as configurações, como:

- Sua infraestrutura deve ser a mesma para a topologia de rede e configuração de firewall.
- Sua configuração de sistema operacional, incluindo os patches deve ser a mesma.
- Seu stack de aplicações deve ser o mesmo.
- Os dados do seu aplicativo devem estar em um estado conhecido e válido, pois, migrar dados quando está sendo realizado melhorias pode causar muitos problemas nas implantações.

Ferramentas como Puppet e InstallShield também podem ser utilizadas junto com um repositório de controle de versão para gerenciar as configurações dos seus ambientes.

Verificando o aprendizado

Questão 1

Observe as seguintes afirmativas em relação aos estágios de implantação da pipeline:

I – O primeiro estágio (commit) é responsável por compilar o código, rodar os testes de unidade, realizar análise do código e criar instaladores.

II – O segundo estágio será iniciado manualmente após a conclusão (com sucesso) do primeiro estágio, sendo esse responsável pelos testes de aceitação automatizados de longa duração.

III – O servidor de implantação contínua deve permitir que você realize esses testes em suítes que podem ser executadas em paralelo.

Estão corretas as afirmativas:

A

I.

B

II.

C

I e II.

D

I e III.

E

Nenhuma das afirmativas.

A alternativa D está correta.

O segundo estágio deve ser iniciado automaticamente após a conclusão (com sucesso) do primeiro estágio. O processo de implantação da pipeline é algo que tem o foco na automatização, a fim de agilizar o processo e evitar erros por ações manuais.

Questão 2

Em relação ao processo de implantação, analise as afirmativas a seguir e marque a opção que contém somente afirmativas corretas:

I – O script de implantação precisa ser diferente para cada ambiente.

II – Uma pipeline eficiente é aquela que tem como um dos objetivos o feedback o mais rápido possível.

III – Um método para garantir que os binários estão intactos é a adição de hashes no momento da criação deles.

IV – Todos os binários que passaram pelo teste de aceitação devem ser exatamente os que chegam à produção.

V – Utilizar o mesmo script de implantação para todos os ambientes aumenta seu controle sobre possíveis problemas de mau funcionamento.

A

II, III, IV, V

B

I, III, IV, V

C

I, II, IV, V

D

I, II, III, V

E

I, II, III, IV

A alternativa A está correta.

Um script de implantação igual para todos os ambientes passa a filtrar os motivos de mau funcionamento, como alguma configuração no arquivo específico do ambiente em que está se trabalhando, um problema na infraestrutura/serviço que a sua aplicação precisa ou a configuração do seu ambiente.

Estágio de commit

O que é commit?

Confira alguns aspectos sobre este assunto.

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

A implantação contínua tem como sua maior vantagem a possibilidade de avançar para uma versão nova, ou retroceder para uma versão antiga, essencialmente sem riscos. Os bugs críticos que podem acontecer por alguma interação não prevista serão detectados pelos testes automatizados, voltando nesse ponto para a versão estável e permitindo que você trabalhe na nova versão off-line.

Porém, para alcançar esse cenário tão almejado, é necessário a automação de uma série de testes, incluindo a automação da implantação em ambientes de teste, de preparação e de produção. O intuito disso tudo é remover totalmente as etapas manuais, que podem ser intensivas e propícias a erros.

Existem inúmeros estágios possíveis para isso, mas os mais comuns nesse caso são:

- Estágio de confirmação
- Estágio de teste de aceitação automatizados
- Estágio de testes manuais

Veja os detalhes de cada estágio:

Confirmação

Estágio responsável pela eliminação de compilações que não estejam aptas para produção. Esse estágio é responsável em otimizar o feedback recebido pela equipe de desenvolvimento, garantindo que o sistema funcione em um nível técnico. Nele, acontece a compilação, análise de código e, principalmente em nível de unidade, testes automatizados.

Testes de aceitação automatizados

Estágio que tem como objetivo garantir que o sistema atenda a todas as necessidades e especificações do cliente, tanto em nível funcional quanto em nível não funcional.

Testes manuais

Estágio que costuma incluir ambientes de testes exploratórios, ambientes de integração e aceitação do usuário teste, confirma que o sistema é utilizável e cumpre com os requisitos solicitados, detectando defeitos que não tiveram flag pelos testes automatizados, e verifica se fornece valor aos seus usuários.

Commit

Estágio no qual é imprescindível a análise de código para garantir que a qualidade esteja dentro dos padrões estabelecidos. Ferramentas como o SonarQube auxiliam com a verificação da complexidade e duplicidade do código.

Caso haja sucesso nessas etapas, é possível utilizar o Apache Maven para gerar os artefatos necessários, enviando-os via script para um servidor de automação e gerenciamento, como o Jenkins, para o repositório de artefatos, onde será possível recuperá-los nos estágios da pipeline seguintes.

Humble e Farley (2010) comentam que há alguns pontos principais que você deve fazer no estágio de commit que, preferencialmente, devem demorar menos de 5 minutos e não mais do que 10 minutos. Esse espaço de tempo é dado porque as tarefas são executadas como um conjunto de trabalhos em uma grid de build (uma facility fornecida pela maioria dos servidores de implantação contínua). Segundo os autores, os passos que devem ser respeitados seriam:

- Compilar o código (se necessário).
- Criar binários para uso em estágios posteriores.
- Realizar análises do código para checar sua estabilidade.
- Preparar artefatos, como testes de data-base, para ser usado nos próximos estágios.

Entenda melhor cada um desses passos:

Passo 1

Compilar a última versão do código fonte e notificar os desenvolvedores, que trabalharam no último check-in com sucesso, para checar qualquer erro na compilação. Essa instância da pipeline é imediatamente descartada, caso seja identificado um erro neste passo.

Passo 2

Efetuar uma sequência de testes, sendo otimizada para ser executada extremamente rápido. Essa sequência de testes pode ser referida como testes de commit ao invés de simplesmente testes de unidade. Pelo fato de serem mesmo testes de unidade, é benéfico utilizar também um conjunto de testes de outros tipos para aumentar a confiabilidade de que a aplicação está trabalhando, se o teste de commit for positivo.

Passo 3

Focar a otimização de todos os testes: comece executando todos os testes e conforme aprenda sobre quais tipos de falha são comuns nas test runs de aceitação, adicione testes específicos para a sua suíte de teste de commit tentar encontrá-los o mais cedo possível. Mesmo com toda a sua estrutura preparada para execução de testes e compilação de código, isso não irá dizer totalmente sobre as características não funcionais da aplicação.

Passo 4

Conseguir feedback em características não funcionais do código, que como capacidade são mais difíceis, mas ainda é possível conseguir feedback nessas características executando ferramentas de análise, por exemplo: testes de cobertura, manutenção ou brechas de segurança.

Humble e Farley (2010) ainda citam que a falha do código em atingir certos parâmetros para essas medidas deve falhar o teste de commit do mesmo jeito que um teste de falha faz, sendo algumas medidas úteis:

- Teste de cobertura (se os testes de commit só cobrem 5% da sua base de código, eles são inúteis).
- Quantidade de código duplicado.
- Complexidade ciclomática.
- Acoplamentos aferentes e eferentes.
- Quantidade de avisos.
- Estilo do código.

A criação de uma assembly implantável do seu código pronta para implantação em qualquer ambiente subsequente é o último passo do estágio de commit, assumindo que todos os testes até então foram bem sucedidos na execução. Esse passo também deve passar pelo commit stage para ser considerado um sucesso no geral, porque tratar a criação de um código executável como critério de sucesso é um jeito simples e eficiente de garantir que o processo de build também está sob influência constante do sistema de implantação contínua.

Pontos a serem observados do estágio de commit

Confira alguns aspectos sobre este assunto.

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

É importante conseguir feedback o mais rápido possível. É uma boa prática esperar até que o estágio de commit da implantação da pipeline seja um sucesso. Caso falhe, você irá então arrumar o problema ou reverter para a versão de controle para voltar com o código antes de ser alterado.

Os desenvolvedores em um “mundo ideal”, onde há largura de banda ilimitada e poder de processamento infinito, provavelmente esperariam todos os testes serem realizados, até os manuais, porque nesse caso seria possível arrumar qualquer problema imediatamente. Mas isso não seria nada prático, já que os últimos passos da pipeline de implantação (testes automatizados de aceitação, teste de capacidade e teste manual de aceitação) são atividades bastante longas.

Na hora de decidir sobre a escolha de um candidato para lançamento, ter sucesso no estágio de commit é extremamente importante porque é um ponto no processo de desenvolvimento que, quando avançado, libera os desenvolvedores para trabalharem na próxima tarefa, ainda sim mantendo a responsabilidade de monitorar o progresso de outros estágios. Mesmo que você apenas implante o processo de estágio de commit no seu processo de desenvolvimento, já fará um passo enorme na qualidade e confiabilidade da entrega das suas equipes.

O repositório de artefatos

Os binários e relatórios que são gerados do estágio de commit precisam ser guardados para serem reutilizados nos estágios mais tardios da sua pipeline, esse lugar se chama repositório de artefatos. É como se fosse um sistema de controle de versão da integração contínua, porém ele só precisa manter algumas versões. A partir do momento em que um candidato a lançamento falhou em algum estágio da pipeline de implantação, não há mais necessidade de manter a versão, podendo então deletar os binários e relatórios do repositório de artefatos.

> É essencial que seja possível fazer um trace back a partir dos seus softwares lançados para as revisões na versão de controle que foram utilizados para criá-lo.

Para que isso seja feito, uma instância da pipeline deve ser correlacionada com as revisões em seu sistema de controle de versão que o deu trigger. Checar qualquer coisa no seu controle de fonte como parte da sua

pipeline faz esse processo introduzir mais revisões associadas com a sua pipeline, tornando ele mais complexo.

O processo de criação de binários ser repetível é um dos critérios para uma boa estratégia de gerenciamento de configuração, ou seja, caso os binários sejam deletados e o estágio de commit da mesma revisão em que foi gerado seja reexecutado, você tem que receber os binários originais novamente.

Saiba mais

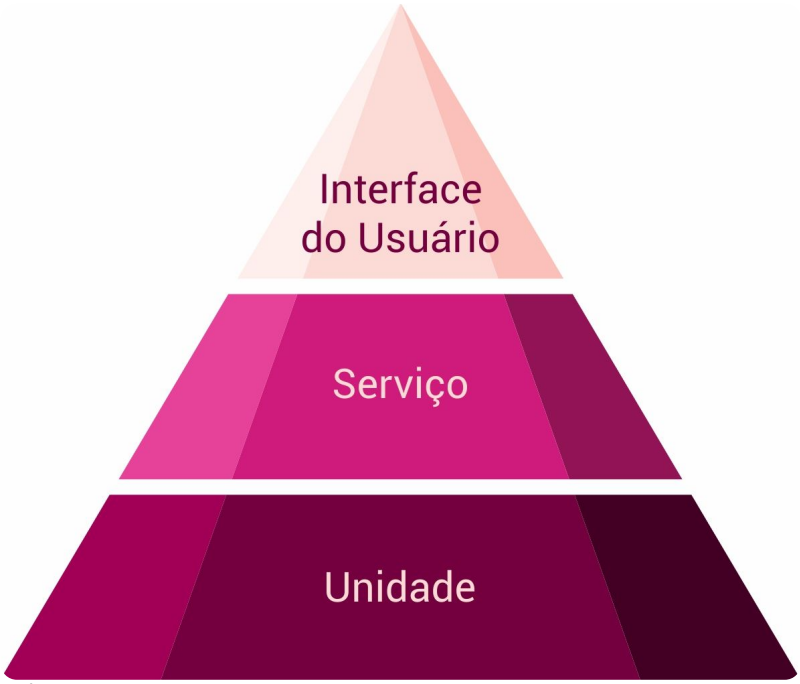
A maioria dos servidores de integração contínua modernos possui um repositório de artefatos, incluindo configurações de quanto tempo os artefatos indesejáveis irão ficar no repositório até serem excluídos, fornecendo mecanismos para especificar quais artefatos manter seguido dos trabalhos a serem executados, podendo até disponibilizar uma interface web para a sua equipe ter acesso a relatórios e binários.

Softwares dedicados para reposição de artefatos como Nexus ou um gerenciador de repositórios no estilo do Marvin, podem ser utilizados alternativamente, pois gerenciadores de repositórios tornam muito mais fácil acessar binários de máquinas de desenvolvimento sem ter que integrar com o seu servidor de integração contínua.

Princípios das suites de teste de commit

A grande maioria dos seus testes de commit precisa ser composta de testes de unidade, pelo fato de que eles trabalham com rapidez tanto em execução quanto feedback, fazendo um cobrimento de boa parte do seu código.

Testes de commit que cobrem menos do que 80% não passam tanta confiabilidade já que possuem muito código que não foi testado, estando sujeito a falhas indesejáveis. Lembrando que todo esse preparo até a implantação contínua é justamente para que não se crie dúvidas de resultados gerados durante o processo. Se um teste de commit passa, ele precisa gerar confiança de que realmente não há chances de algo falhar.



Pirâmide de teste automatizado.

Cohn (2009) elaborou uma pirâmide de como você deve estruturar a sua suíte de testes automatizados, onde os testes de unidade compõem a maioria dos testes, mas não deve demorar mais do que alguns minutos por conta da rapidez de sua execução. Em seguida, há menos testes de aceitação (divididos em testes de serviço e de interface de usuário) que possuem um tempo mais elevado de execução, mas todos os níveis da pirâmide são importantes para um teste automatizado efetivo.

Verificando o aprendizado

Questão 1

No estágio de commit, é imprescindível a análise de código para garantir que a qualidade esteja dentro dos padrões estabelecidos, gerando artefatos e encaminhando-os para um servidor de automação e gerenciamento onde posteriormente será possível recuperá-los caso seja necessário.

Sobre os passos que devem ser respeitados no estágio de commit, está correto:

A

Não criar binários, pois eles ocupam muito espaço.

B

Usar fonte número 11.

C

Armazenar os artefatos e binários gerados em um disquete.

D

Armazenar a versão do código fonte mesmo quando for detectado um erro.

E

Realizar análises do código para checar sua estabilidade.

A alternativa E está correta.

A realização de análises do código para checar sua estabilidade é um passo importante no estágio de commit, pois a confiabilidade de um sistema automatizado é o que irá eliminar suspeitas do próprio sistema caso algum erro seja detectado.

Questão 2

Sobre os binários e relatórios no estágio de commit, assinale a alternativa que apresenta a sequência correta.

( ) A criação de binários ser repetível é um dos critérios para uma boa estratégia de gerenciamento de configuração.

( ) Binários e relatórios que são gerados do estágio de commit precisam ser guardados em um repositório de artefatos.

( ) O repositório de artefatos é o equivalente a um sistema de controle de versão da integração contínua, armazenando todas as versões.

A

V – V – V

B

V – F – V

C

V – V – F

D

V – F – F

E

F – F – V

A alternativa C está correta.

O repositório de artefatos mantém apenas algumas das versões, descartando os binários e relatórios que falharam em algum estágio da pipeline de implantação.

Testes de aceitação

Implantando testes de aceitação

Confira os fundamentos do processo da implantação contínua e como este processo garante agilidade e qualidade ao sistema implantado.

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Não é raro acontecer que uma aplicação tenha todos os testes de unidade feitos com sucesso, mas não atenda aos requisitos para os quais ela foi desenvolvida. É porque mesmo sendo uma parte essencial de qualquer desenvolvimento baseado na integração contínua, os testes de unidade por si só não são suficientes.

Wolff (2017) explica que uma diferença entre os testes de unidade e de aceitação não está na tecnologia sendo usada, porque testes de aceitação também podem ser implementados em frameworks de teste de unidade como JUnit.

Testes de unidade

São ferramentas para os desenvolvedores, escrito por eles mesmos para garantir a implementação correta, avaliando unidades individuais de uma aplicação em isolamento e o usuário talvez nem saiba que há testes de unidade (O primeiro detalhe a se observar são os stakeholders).

Testes de aceitação

São usados por desenvolvedores e usuários para garantir a implementação correta dos requisitos em relação à lógica do domínio, avaliando grandes partes do sistema, podendo ser até o sistema em um todo, tendo como objetivo ser uma lógica de domínio salvaguarda.

Além dos testes de unidade e as várias categorias de testes de commit, Farley (2007) recomenda testes automatizados de aceitação para garantir a maior eficiência da aplicação. Os testes de aceitação são testes funcionais que trabalham o sistema de ponta a ponta, com exceção de sistemas que estão externos e fora do nosso controle, onde são controlados por sistemas de monitoramento de implantação contínua, como o CruiseControl em ambientes dedicados para essa tarefa.

Pode-se dizer que o teste de aceitação automatizado é o segundo marco no ciclo de vida de um candidato a lançamento, pois a pipeline só irá autorizar os estágios subsequentes como implantações manuais solicitadas após a build passar pelo teste de aceitação automatizado.

> O objetivo dele é garantir que o sistema entregará a qualidade que o cliente está esperando e que irá atender ao critério de aceitação, porém ele também pode ser usado como uma suíte de teste de regressão, podendo verificar se nenhum bug foi introduzido pelas mudanças.

Como parte do processo normal de desenvolvimento, é necessário atribuir a resposta imediata a flags do teste de aceitação automatizado, pois identificar se a flag foi resultado de uma regressão que foi introduzida, uma mudança intencional no comportamento da aplicação ou um problema com o teste é crucial para a eficiência da metodologia.

Mesmo que seja possível passar por cima do teste de aceitação, ele por si só possui tantos critérios que será gasto muito mais tempo tentando ignorar o problema do que tentar resolvê-lo, então o próprio teste de aceitação acaba também garantindo que a equipe sempre faça a coisa certa.

> Uma das desvantagens do teste de aceitação automatizado seria, quando feito errado, ter um custo mais elevado para manter e isso seria uma percepção errada.
>
> —
>
> (HUMBLE e FARLEY, 2010)

Porém, é possível diminuir o custo desse teste automatizado até ele ser eficiente em seu custo aplicando-o em todas as builds que passam os testes de commit, surtindo efeito dramático no processo de entrega do software. Como o loop de feedback é menor, os defeitos são encontrados cedo, reduzindo o custo para reparo e com os desenvolvedores, testers e clientes precisam trabalhar com uma colaboração muito maior entre eles, todos estarão focados no valor de negócio esperado que a aplicação entregue.

Testes de aceitação utilizando a GUI da aplicação

Testando a interface gráfica, qual a relação com os testes de aceitação

Confira alguns testes de aceitação utilizando a GUI da aplicação.

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

A utilização de teste de aceitação utilizando a interface gráfica (GUI da aplicação - Graphic User Interface) do usuário não é uma prática muito recomendada na implantação contínua. Isso porque a natureza do desenvolvimento da aplicação é a constante evolução e, conforme for seguindo com o cronograma do projeto, há grande possibilidade das equipes de engenheiros, testers, usuários etc. não conseguirem colaborar simultaneamente por conta da complexibilidade. Essa falta de colaboração simultânea vai contra os fundamentos da implantação contínua.

Saiba mais

Como os testes de aceitação automatizados são feitos para testar justamente a funcionalidade e simular as interações dos usuários com o sistema, seria lógico implantar diretamente na GUI, porque caso contrário não estaríamos testando o mesmo caminho de código que os usuários vão utilizar em interações reais.

Porém, pela rápida taxa de mudança da interface ao longo de todo o projeto, leves mudanças na GUI podem acabar quebrando as suites de aceitação caso seus testes estejam linkados diretamente à GUI, não sendo expressamente direcionado à estrutura de desenvolvimento, mas também sendo possível quando houver melhorias de usabilidade e correções de gramática.

> Para evitar esse problema, basta agregar abstrações diretamente nos testes da GUI que são abstratos da interface concreta. Dessa forma, se um botão for renomeado ou movido, apenas a layer abstrata terá que ser adaptada e o teste mantém sua forma inalterada. Mas, mesmo assim, isso só se aplica a testes simples de GUI pelo fato de eles serem mais fáceis de entender, por conta de apenas automatizarem a interação com a interface do usuário.
>
> —
>
> (WOLFF, 2017)

No caso da adição de mais abstrações, os testes passam a ser cada vez mais complexos e se encontram os mesmos problemas de antes: a colaboração simultânea passa a ser dificultada e a fluidez da implantação contínua deixa de existir.

Apesar de suas limitações, os testes de interface gráfica do usuário são ferramentas extremamente úteis pelo fato de o trabalho manual ser automatizado (a fundação da implantação contínua) e como o resultado é facilmente entendido, durante qualquer tempo e hora, os usuários conseguem checar exatamente o que está acontecendo com a GUI.

Criação da automação de testes de aceitação

Executando testes de aceitação automaticamente

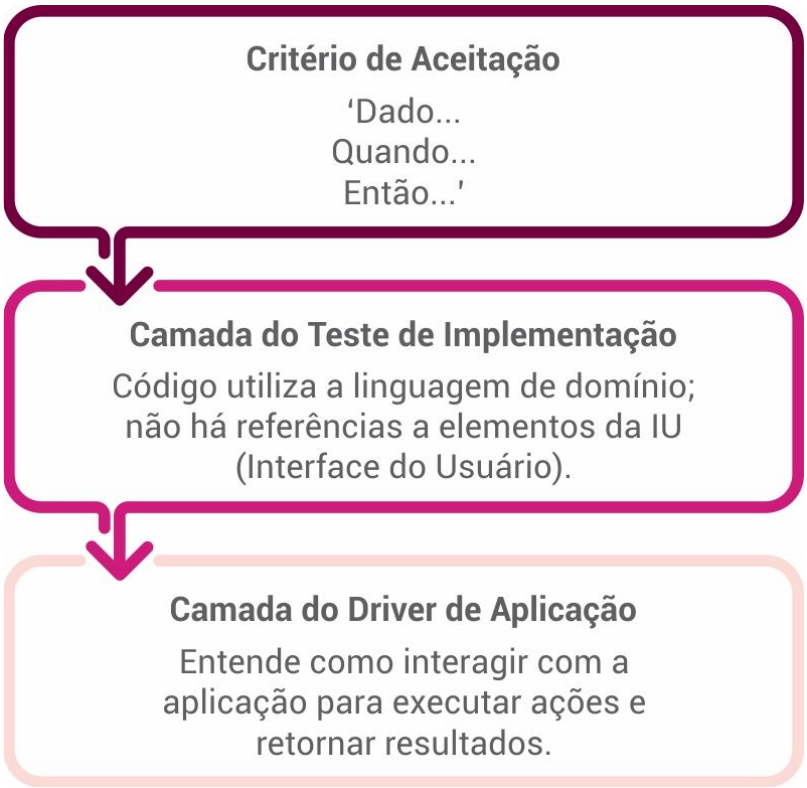
Entenda melhor sobre a criação da automação de testes de aceitação.

> Conteúdo interativo
>
> Acesse a versão digital para assistir ao vídeo.

Testes de aceitação precisam ser derivados de um critério de aceitação, então sempre será necessário que seu critério seja escrito com a automação e o princípio **INVEST** em mente.

> INVEST
>
> Vem do inglês de Independent, negotiable, valuable, estimable, small e testable, ou seja, independente, negociável, valioso, estimável, pequeno e testável.

Seguindo esse princípio, é possível canalizar a direção do seu teste de aceitação através de vários requisitos, a fim de facilitar a manutenção do teste, pois ele será muito mais fácil de entender por conta da funcionalidade estar bem expressa. Após determinar todo o seu critério de aceitação, descrevendo o que será entregue ao usuário, então o próximo passo será a automação desse teste, que no caso deve sempre ser por camadas.



Camadas da automação dos testes de aceitação.

O primeiro fundamento da criação da automação será o critério de aceitação. Algumas ferramentas como Cucumber permitem você colocar o critério de aceitação direto nos testes e linká-los nas implementações subjacentes ou também agregar o critério de aceitação nos nomes do seu teste xUnit, executando-os diretamente da framework do xUnit. Como dito, testes de implementação utilizando a GUI não são confiáveis, podendo quebrar apenas com mudanças sutis na interface.

Testes de aceitação precisam de cuidado para serem mantidos ao longo do tempo. Eles devem sempre chamar camadas subjacentes, chamadas de camadas de aplicação do driver, essas layers possuem uma API (Application Programming Interface, ou interface de programação de aplicação) justamente porque ela saberá como executar as ações e devolver os resultados.

Criando os testes de aceitação

Um projeto deve ter uma equipe de desenvolvedores, analistas e testers, porém, isso pode acabar não sendo um caso e desenvolvedores acabam realizando também o trabalho de analistas ou analistas também agindo como testers. Em uma situação ideal, o cliente estará junto com a equipe contribuindo como analista também, porém o mais importante é que todas essas funções existam na equipe, mesmo que haja casos de não existir equipes separadas para essas funções.

Saiba mais

Sobre as funções dentro do projeto, testers são fundamentais. Sua função é garantir que a qualidade do software sendo desenvolvido seja compreendida por todos, desde o cliente até a equipe de desenvolvimento.

Já os analistas são responsáveis pela representação do cliente e usuários do sistema, trabalhando junto com os testers para que os critérios de aceitação sejam propriamente especificados, garantindo que a funcionalidade programada entregará o desejado por meio da ponte feita entre os desenvolvedores e as necessidades do cliente.

Testes de aceitação devem sempre estar de acordo com a aplicação e entregar qualidade para os usuários, sendo essa a função do analista: a de compreender, definir e elaborar o critério de aceitação.

Humble e Farley (2010) citam o exemplo de Chris Matts e Dan North sobre um critério de aceitação: **"Dado** um contexto inicial"; **"Quando** um evento ocorre"; **"Então** há alguns resultados". Esse teste de aceitação tem o objetivo de começar a sua aplicação **dada** uma função para executar as ações **quando** a interação for completa e **então** verificar o estado quando terminado.

- **Dado**: representa o estado da sua aplicação no começo do teste.
- **Quando**: descreve a interação entre o usuário e a aplicação
- **Então**: é usado para o estado da aplicação depois que essa interação for completa.

O critério de aceitação pode ser exemplificado como:

Função: Efetuar uma compra
Cenário: Compra deve debitar do usuário corretamente.
Dado que exista um instrumento chamado bond
E exista um usuário chamado Jonas com 80 reais na sua conta
Quando eu logo como Jonas
E eu seleciono o instrumento bond
E eu executo uma ordem de comprar 3 custando 20 reais cada
E a ordem é efetuada com sucesso
Então eu tenho 20 reais restantes na minha conta

Verificando o aprendizado

Questão 1

Os testes de aceitação do software analisam a funcionalidade principal do sistema. A prática de utilizar o teste de aceitação lincados diretamente na interface gráfica do usuário não é recomendada porque

A

o teste pode quebrar com qualquer mudança na interface gráfica do usuário.

B

o teste precisa ser adequado à interface gráfica do usuário.

C

o teste não evolui junto com a interface gráfica do usuário.

D

o teste pode quebrar com a formatação da interface gráfica do usuário.

E

o teste não é necessário nessa etapa.

A alternativa A está correta.

Como a interface gráfica do usuário sofre rápidas mudanças ao longo do desenvolvimento, qualquer mudança, mesmo que seja uma simples correção de gramática, é passível de quebrar o teste de aceitação caso ele seja lincado diretamente à interface gráfica do usuário.

Questão 2

Sobre os critérios de aceitação, assinale a alternativa correspondente às afirmações a seguir.

I – Representa o estado da sua aplicação no começo do teste.
II – É usado para o estado da aplicação depois que essa interação for completa.
III – Descreve a interação entre o usuário e a aplicação.

A

Dado – Quando – Então

B

Então – Quando – Dado

C

Dado – Então – Quando

D

Então – Dado – Quando

E

Quando – Dado – Então

A alternativa C está correta.

Os critérios de aceitação de um teste têm o objetivo de começar a sua aplicação, dada uma função para executar as ações, quando a interação for completa e então verificar o estado quando terminado.

Fundamentos da implantação contínua

Implantando testes de aceitação

Confira os fundamentos da integração contínua, correlacionando com as outras fases do processo DevOps.

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Para uma implantação contínua ser feita com sucesso, é necessário trabalhar em uma fundação sólida de integração e entrega contínua, pois sua função é justamente levar ambas as metodologias à sua conclusão lógica. Ou seja, toda a estrutura de compilação automatizada e teste, além do processo e os fluxos de trabalho para verificação contínua do código. Veja:

1

Integração contínua

Responsável pela execução de etapas que normalmente aconteceriam na fase de “integração” ao longo do processo de desenvolvimento, gerando como vantagem a análise de bugs sem ter que esperar o código inteiro estar completo para identificar possíveis problemas. Isso faz com que muito tempo seja poupado e aumente a eficiência da empresa.

2

Entrega contínua

Responsável por automatizar as builds e testes previamente estabelecidos na integração contínua. Diferente do método tradicional, no qual o produto ia do desenvolvedor para os testadores e depois para o gerente de lançamento, essa etapa tem como objetivo fazer com que a equipe tenha controle de todas as etapas de construção, testes e lançamento. Dessa forma, o processo é acelerado e passa a ter mais confiabilidade por conta de maior estabilidade e menor risco de erros.

Como essas metodologias são recentes, a Implantação Contínua acaba sendo apresentada como um sinônimo da Entrega Contínua. Isto é um erro, porque apesar de o objetivo ser o mesmo (a entrega de atualizações e features para o usuário), a Implantação Contínua se dá pela automação total do pipeline, enquanto a Entrega Contínua prevê a utilização de um lançamento manual do software.

Todo esse processo é bastante delicado e é um grande desafio para muitas empresas, pois ele precisa ser à prova de falhas. O esforço inicial para criar uma fundação sólida da integração contínua envolve o suporte de desenvolvedores, testadores e engenheiros de build.

Saiba mais

Outro detalhe a ser observado é que todo esse processo deve ser implementado de forma gradual, fornecendo incentivos com o objetivo de todos os funcionários estarem alinhados e garantir a adesão das equipes à integração contínua.

Inicialmente, é necessário adotar processos de integração e implantação automáticos com códigos funcionantes, sem congelamento, além de traçar estratégias de testagem que elenque uma quantidade limitada de profissionais, fragmentando o processo. Assim, evita-se a ansiedade de toda a empresa durante um lançamento.

Outro ponto é a não-idealização de um processo perfeito, sendo imprescindível a manutenção de uma equipe capacitada e um ambiente voltado para implementar a metodologia DevOps.

Executando a implantação contínua

Entenda mais sobre implantação contínua.

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Os estágios mais importantes da pipeline são os últimos: implantação em ambientes de teste e produção, onde nesse ponto não há mais testes automatizados a serem feitos, mas continuam sendo uma parte integral da sua pipeline. Nesse ponto, sua Implantação Contínua tem que ser capaz de implantar qualquer versão que a sua aplicação passou com sucesso pelos testes automatizados, com todos da equipe podendo visualizar quais mudanças foram feitas naquele momento.

Saiba mais

O melhor jeito de reduzir o risco de lançar sua aplicação e encontrar erros inesperados é a automação da criação de um novo ambiente de teste para cada lançamento, atualizando ambientes preexistentes automaticamente, para poder garantir que todos os testes automatizados estão sendo feitos corretamente.

O processo de preparar ambientes para implantação e gerenciamento após o lançamento precisa estar baseado na ideia de que o estado desejado da sua infraestrutura tem que ser especificado pela configuração por tipo de versão, sendo ela totalmente autônoma e corrigir ela mesma para a versão desejada, permitindo que você sempre tenha conhecimento do estado da sua infraestrutura por ferramentas de monitoramento.

> A Implantação Contínua, feita sendo pensada desde o começo de um projeto, é uma ótima maneira de garantir que todo o processo seja feito da maneira correta.

Não é possível fazer uma Implantação Contínua sem automatizar todo os seus testes de sistema em um ambiente o mais próximo possível da produção ou até o processo de build, deploy, teste e lançamento. Mesmo que não seja possível implantar todos os conjuntos de mudanças que passam por todos os testes, é sim uma boa prática pensar desde o começo em fazer um processo que permita isso caso seja necessário.

Dessa forma, é possível reduzir a periodicidade de lançamentos, que antes se dava semanalmente, para lançamentos a cada hora, por exemplo, dependendo das necessidades do aplicativo/empresa. Para uma quantidade cada vez maior de setores da indústria, a capacidade de entregar recursos rapidamente e responder de forma ágil ao feedback dos usuários tornaram-se essenciais.

Vamos analisar mais alguns pontos a seguir:

Velocidade e qualidade

Durante o planejamento da Implantação Contínua, a equipe precisa considerar como as suas alterações serão lançadas. Pois, uma vez que a implantação está intrinsecamente ligada a um processo totalmente automatizado, é preciso garantir também que a velocidade não prejudique a qualidade.

Atualizações

Por outro lado, também deve-se escolher atualizações que entrem em execução sem interromper os serviços on-line, para que a distribuição seja uma extensão do processo de testes automatizados.

Tipos de implantação

Implantação canário ou azul/verde

O video a seguir vamos entender mais sobre implantação canário ou azul/verde.

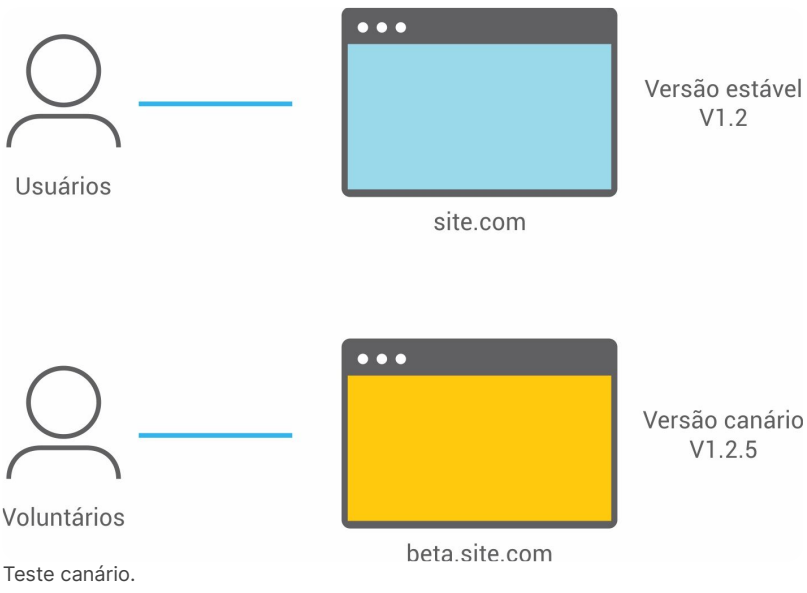
Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Implantação canário

Uma implantação canário (canary deployment) é uma versão inicial da aplicação dividida em duas versões: a estável e a em desenvolvimento. Com essa metodologia, é possível publicar uma versão em desenvolvimento para uma pequena porcentagem dos usuários.

Há casos em que essa pequena população é escolhida de forma involuntária por sorteio, região ou de forma voluntária, onde a versão canário é disponibilizada para os usuários testem a aplicação.



Dessa forma, você consegue monitorar o comportamento e as métricas de uso, verificando se a nova versão não introduziu novas falhas antes de distribuí-la de forma mais abrangente.

Conforme o desenvolvimento da versão canário procede, mais usuários vão sendo escolhidos para a transição de versão gradativamente até então a aplicação se tornar a nova versão estável. A implantação canário possui muitos benefícios, como:

Teste A/B

São apresentadas duas alternativas do mesmo aplicativo para os usuários, sendo então determinado qual possui melhor recepção.

Teste de capacidade

Já são incorporados na implantação canário. Qualquer problema de performance pode ser identificado conforme mais usuários vão sendo migrados para a versão canário.

Feedback

Você consegue receber feedback diretamente de usuários reais.

Rollback imediato

Qualquer problema na implementação de alguma alteração do código é facilmente recuperada por meio do rollback para versões anteriores.

Sem tempo de inatividade

Assim como na implantação azul/verde, a implantação canário não tem tempo de inatividade durante a atualização no ambiente de produção.

Vale ressaltar que é necessário ter uma análise prévia antes de começar a publicar uma versão canário, como: duração de execução da aplicação canário, como os usuários serão selecionados (aleatórios, por região, por registro prévio etc). Também é uma boa prática oferecer a versão canário para usuários internos e empregados antes de começar a implantação canário e converter os usuários para a nova versão.

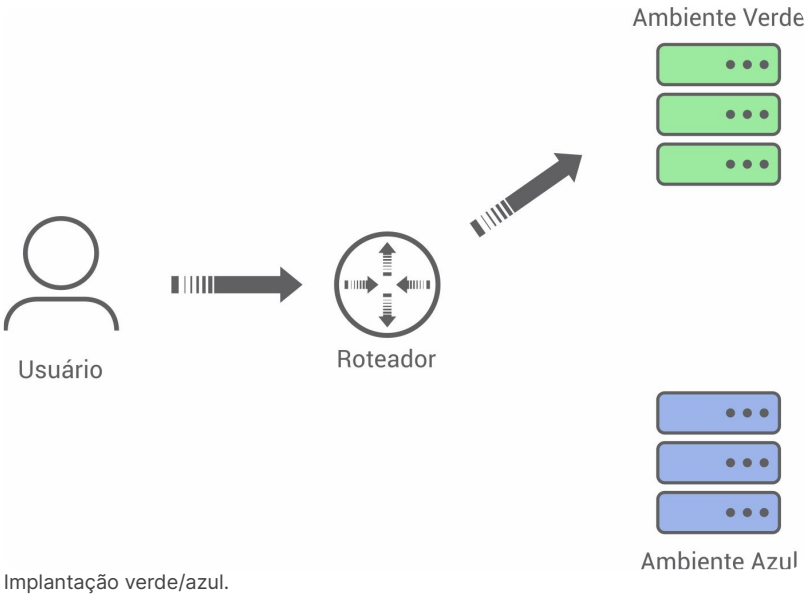
Implantação azul/verde

Um processo de implantação azul/verde é uma técnica comum para organizações que praticam implantação contínua, já que torna mais fácil reverter uma versão caso haja algum problema, mantendo o código antigo on-line até que você tenha certeza de que as alterações estejam funcionando conforme o esperado. Se necessário, você pode seguir uma implantação canário inicial com um lançamento azul/verde.

Esse modelo, assim como a implantação canário, também é executado em ambiente de produção sem tempo de inatividade. Isso se dá pelo fato da transferência gradual de usuários de uma versão anterior (ambiente azul) para a nova (ambiente verde). Após o tráfego ser totalmente transferido do ambiente azul pro verde, é comum passar a utilizar o ambiente azul como um ambiente de reversão (rollback).

Isso se dá de duas formas:

1. apontando o balanceador de carga da versão antiga para a nova;
2. pelo rotacionamento de DNS entre as duas versões.



Implantação verde/azul.

Uma plataforma que se adequa às necessidades da implantação azul/verde é a Kubernetes (ou k8s), uma plataforma open source que automatiza as operações dos containers Linux, com objetivo de eliminar grande parte dos processos manuais necessários para implantar e escalar as aplicações em implantações contínuas.

Se você está executando uma implantação azul/verde ou implantando substituições diretas, monitorar a integridade do sistema de produção é essencial caso deseje ser capaz de responder rapidamente a quaisquer bugs que tenham escapado ao processo de lançamento.

Ficar de olho nas métricas específicas que indicam a integridade do seu sistema, desde espaço em disco e uso da CPU até o número de solicitações ou transações, e compará-los com uma linha de referência pode fornecer um aviso prévio quando as coisas não estiverem se comportando como deveriam. Você pode então decidir se deseja reverter a alteração ou avançar, introduzindo uma correção através do pipeline.

Verificando o aprendizado

Questão 1

Como são metodologias recentes, a entrega contínua e implantação contínua costumam ser apresentados como sinônimos. Contudo, isso é um erro porque

A

a implantação contínua é a automação total da pipeline, já a entrega contínua é responsável pela entrega dos artefatos e binários ao repositório de artefatos.

B

a implantação contínua prevê a utilização de um lançamento manual do software, já a entrega contínua é a automação total da pipeline.

C

a implantação contínua é a automação total da pipeline, já a entrega contínua prevê a utilização de um lançamento manual do software.

D

a implantação contínua prevê a utilização de um lançamento manual do software, já a entrega contínua é responsável pela entrega dos artefatos e binários ao repositório de artefatos.

E

a implantação contínua é responsável pela entrega dos artefatos e binários ao repositório de artefatos, já a entrega contínua prevê a utilização de um lançamento manual do software.

A alternativa C está correta.

A implantação contínua tem como objetivo a automação total da pipeline para acelerar o lançamento de aplicativos, o que antes, pela entrega contínua, era automatizado até o ponto de implantação do software em ambientes de teste ou produção.

Questão 2

Sobre os benefícios da implantação canário (Canary Deployment), assinale a alternativa que contém somente características corretas.

I – Teste A/B
II – Teste de capacidade
III – Feedback
IV – Rollback imediato
V – Tempo de inatividade curto

A

I, II, III, V

B

I, II, IV, V

C

II, III, IV, V

D

I, III, IV, V

E

I, II, III, IV

A alternativa E está correta.

A implantação canário não possui tempo de inatividade, sendo possível atualizar o ambiente de produção em tempo real.

Considerações finais

Apesar de complexa para ser implementada, requerendo uma mudança comportamental da equipe e colaboração entre todas as partes durante todo o processo de desenvolvimento, percebemos que a implantação contínua é mais uma metodologia elaborada com o intuito de otimizar ainda mais o processo da publicação de aplicações, aumentando, por meio da automação, a confiabilidade durante todo o pipeline de implantação, reduzindo a probabilidade de erro humano no projeto.

A implantação contínua é o aprimoramento de uma metodologia que já era considerada bastante eficiente, unindo a integração contínua e a entrega contínua que, até então, tinham como resultado os testes de ambiente de produção manuais à automação de todo o processo até a implantação.

Com isso, além de ter um ponto de retorno em todos os pontos do desenvolvimento, caso algum teste dê uma flag de problema, é possível ter todo um controle na hora de aplicar uma atualização no ambiente de produção graças à gradualidade das implantações. Os usuários podem ir recebendo a atualização conforme for decidido na estratégia de entrega e caso algo inesperado aconteça, é totalmente reversível e sem nenhum downtime.

Podcast

Para encerrar, ouça um bate-papo sobre o processo de implantação contínua.

Conteúdo interativo

Acesse a versão digital para ouvir o áudio.

Explore +

Confira a indicação que separamos especialmente para você!

Leia o livro **Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation**, de Jez Humble e David Farley, que tentam abranger o máximo possível de conhecimento que possa ser aplicado a uma empresa, sem criar tendência para o leitor com uma linguagem de programação específica, além de contar com exemplos de boas práticas a serem tomadas durante todo o processo.

Referências

COHN, M. **Succeeding with Agile**: Software Development Using Scrum. Nova York: Addison-Wesley, 2009.

ENTREGA Contínua vs Implantação Contínua. s.d. Microsoft.

INTEGRAÇÃO Contínua. Atlassian, s.d.

IMPLANTAÇÃO Azul/Verde. Redhat, 8 jan. 2019.

FARLEY, D. **The Deployment Pipeline**. ThoughtWorks Studios, 2007.

HUMBLE, J.; FARLEY, D. **Continuous Delivery**: Reliable Software Releases through Build, Test, and Deployment Automation. Nova York: Addison-Wesley, 2010.

PITTET, S. **Saiba mais sobre implementação contínua com o Bitbucket Pipelines**. Atlassian. S.d.

O que é CI/CD? Redhat, 11 maio 2022.

O que é Entrega Contínua? Redhat, 24 nov. 2020.

WOLFF, E. **A Practical Guide to Continuous Delivery**. Nova York: Addison-Wesley, 2017.