Introdução a Práticas DevOps

Marco Mendes
masmendes@pucminas.br
Junho de 2021

1.1 A cultura DevOps

A cultura DevOps tem se estabelecido nos últimos anos para transformar a forma como desenvolvedores e times de operações desenvolvem e mantem suas aplicações. Esta cultura é um forte aliado das práticas arquiteturais, pois busca conectar times de desenvolvimento, qualidade e operações com o uso de práticas e aceleradores de automação de ciclo de vida e monitoração de aplicações.

Uma boa arquitetura deve endereçar atributos de qualidade que são críticos para o negócio. Nos tempos atuais é crítico reduzir o tempo de ciclo no desenvolvimento de aplicações, o tempo para recuperação de incidentes em produção e o esforço gasto em defeitos e retrabalhos. Em linguagem arquitetural, estas organizações estão buscando trabalhar condutores arquiteturais tais como manutenibilidade, testabilidade, implantabilidade, configurabilidade e recuperabilidade.

Em muitas empresas existem muitas dificuldades para que uma aplicação seja operacionalizada com rapidez e estabilidade para ambientes de produção. Nestas empresas, os times de desenvolvimento, qualidade e operações estão dispostos como silos e não mantém uma comunicação frequente e eficaz. Também vemos nestas empresas ciclos longos de entrega (trimestres, semestres ou até anos) e um alto índice de retrabalho em seus produtos.

Nos últimos anos, uma cultura de desenvolvimento que envolve pessoas, práticas e produtos emergiu sob o termo **DevOps**. Esta cultura de desenvolvimento tem suas raízes nos princípios Lean, práticas ágeis de desenvolvimento, processos ágeis de desenvolvimento com o XP (*Extreme Programming*) e melhores práticas de corpos de conhecimento com o ITIL. Embora o termo DevOps ainda represente um conjunto difuso de práticas técnicas e culturais, é também verdade que um conjunto comum de práticas tem trazido notáveis resultados de negócio para muitas organizações. Empresas como WalMart, Staples, Amazon, Uber, Netflix, Microsoft, IBM, Globo.com e Leroy Merlin, entre diversas outras, já possuem casos publicados do valor de negócio da adoção do DevOps dentro de suas TIs.

O valor de negócio do DevOps pode ser expresso em métricas como:

- MTTD (*Mean Time to Deliver*). Também chamado de tempo de entrega na comunidade Kanban, ela mede o tempo gasto desde o momento do nascimento da demanda ou projeto até a sua disponibilização no ambiente de produção para os seus usuários.

- MTTR (*Mean Time to Recover*). Mede o tempo gasto pelo time de operações para recuperar o ambiente de produção de um incidente.
- % de Retrabalho. Mede o percentual do esforço do projeto gasto com atividades não planejadas e resolução de defeitos.

Segundo o relatório *The State of DevOps Report 2016*[1], empresas de alta maturidade em práticas DevOps tem o tempo de entrega de aplicações (MTTD) acelerado em 2500 vezes, são 24 vezes mais eficientes para recuperar falhas (MTTR) e tem 3 vezes menos retrabalhos que empresas de baixa maturidade. Esse relatório cita o caso da Amazon, que hoje realiza 80 implantações por dia em ambiente de produção. O relatório *The Value of IT Automation* do *Gartner Group*[2] mostra como empresas como *Walmart*, e *Staples* aumentaram a eficiência operacional de suas TIs com o uso de práticas DevOps.

Podemos pensar através na cultura DevOps com a confluência de três fatores críticos no desenvolvimento e manutenção de software: pessoas, processos e produtos.

- **Pessoas:** Envolve aproximar de times que trabalharam separados (Desenvolvimento, Qualidade e Operações). A cultura DevOps coloca essas pessoas no mesmo compasso, trabalhando juntas e com o objetivo de garantir ritmo nas entregas e aumentar o fluxo de valor da TI para as áreas de negócio.
- **Processos**: Envolve enxugar a burocracia e desperdícios nos processos tradicionais de fazer e manter software. A cultura DevOps traz as práticas Agile/Scrum e Lean para dentro do ciclo de montagem de arquiteturas e produtos de forma pragmática e acionável para os times de desenvolvimento, qualidade e produção.
- **Produtos:** Envolve usar ferramentas de ciclo de vida para enlaçar disciplinas importantes tais como qualidade contínua, gestão de configuração, automação de testes, gestão de builds, gestão de releases e infraestrutura como código dentro de processos simples e acionáveis. O Azure DevOps, IBM Jazz, GitLab, Puppet Enterprise, Chef ou Atlassian Bamboo são alguns exemplos destes produtos.

Nos ambientes competitivos atuais, as empresas não irão escolher se vão implantar as práticas DevOps, mas quando irão fazê-lo e em qual velocidade.

1.2 DevOps para Criar Progresso Real nos Projetos

A abordagem tradicional de gerenciamento de desenvolvimento de software adia para os momentos finais a verificação se uma funcionalidade está funcionando. Uma funcionalidade **pronta** deveria atender um conjunto significativo de critérios tais como:

[1] https://puppet.com/resources/white-paper/2016-state-of-devops-report
[2] https://puppet.com/resources/analyst-report/value-of-it-automation

- Operar no ambiente real (ou de homologação) do cliente;
- Operar em uma base de dados real (ou similar) a do cliente;
- Operar de forma integrada com os sistemas legados que ele deve conversar;
- Possui uma suíte de testes de automação para garantir bons testes de regressão nos aspectos funcionais e não-funcionais;
- Não gera débito técnico de manutenibilidade no código fonte;

Ao mesmo tempo, defeitos são inerentes no trabalho de se fazer software, que está sujeito a variabilidade do trabalho intelectual humano. E se o **critério de pronto** não é forte, o software irá acumular defeitos ao longo do seu ciclo de vida. E a consequência é que o % de progresso real é na prática menor que o % de progresso declarado em um cronograma desconectado da realidade. Quanto mais robusto o critério de pronto, mais sólido tende a se tornar o produto e maior será a medição real do progresso do produto. Quando times, por imaturidade ou preguiça estabelecem um critério de pronto fraco ou não cumprem o acordo de pronto, haverá trabalho não feito no produto. O trabalho não feito é a diferença entre o trabalho necessário para ir para produção e o trabalho realizado no projeto. Em muitas empresas, o trabalho não feito é tão grande que ele gera um enorme débito técnico no sistema, que se materializa com defeitos em homologação e produção.

A questão que surge é se teremos uma mecânica de trabalho que irá expor e resolver de imediato os defeitos ou se deixaremos que os defeitos permaneçam no sistema e gerem problemas na homologação, ou pior, no ambiente de produção. Times DevOps acreditam na primeira alternativa e para isso estabelecem uma cultura que não apenas reduz a fricção de desenvolvimento, como também garantem a entrega de produtos robustos em produção.

Para isso é necessário que o time estabeleça um critério de pronto sólido para cada funcionalidade. Por exemplo, um **critério de pronto sólido** poderia estabelecer que uma funcionalidade está pronta se ela foi:

- compilada em um ambiente limpo, diferente da máquina de desenvolvimento onde ela foi codificada;
- testada com uma suíte de testes de unidade;
- testada com robôs de automação de testes de telas;
- testada com robôs para automação de testes de segurança, usabilidade, performance, escalabilidade ou recuperabilidade;
- integrada com sucesso no tronco principal;
- montada em um build com um número de versão;
- promovida de forma automatizada para um ambiente de homologação;
- aprovada para ser colocada em produção após os testes exploratórios e sistêmicos do time de QA.

A cultura DevOps busca apoiar o estabelecimento de critérios de prontos sólido, através de práticas culturais, práticas técnicas e suporte de aceleradores de automação. Embora o restante do capítulo seja dedicado a práticas e ferramentas, é relevante reforçar que o primeiro pilar da cultura DevOps está nas pessoas. Se o time não se comprometer com a

mudança cultural, não haverá sucesso ao introduzirmos uma sofisticada ferramenta de gestão de builds ou gestão de releases. Busque isso antes de instalar a configurar a sua primeira ferramenta DevOps.

1.3 Os três caminhos do DevOps

Figura 1: O primeiro caminho DevOps

O **primeiro caminho** permite um fluxo rápido de trabalho da esquerda para a direita de Desenvolvimento para Operações para o cliente. Para maximizar o fluxo, precisamos tornar o trabalho visível, reduzir o tamanho dos lotes e os intervalos de trabalho, aumentar a qualidade evitando que os defeitos sejam passados para os centros de trabalho mais à direita e otimizar constantemente as metas globais. As práticas resultantes incluem processos contínuos de construção, integração, teste e implantação; criando ambientes sob demanda; limitando o trabalho em progresso (WIP); e construir sistemas e organizações que sejam seguros para mudar.

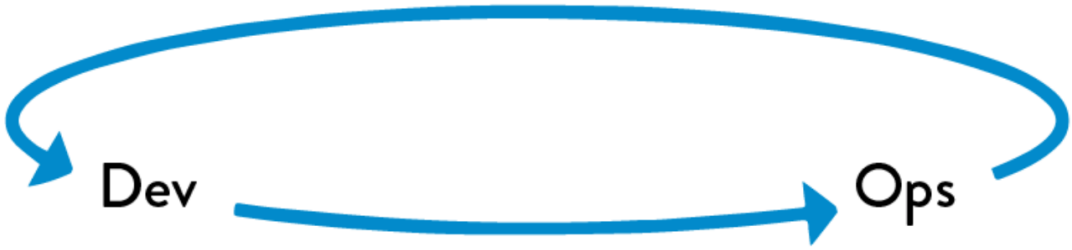


Figura 2: O segundo caminho DevOps

O segundo caminho permite o fluxo rápido e constante de feedback da direita para a esquerda em todos os estágios do fluxo de valor. Isso exige que amplifiquemos o feedback para evitar que os problemas aconteçam novamente ou para permitir uma detecção e recuperação mais rápidas. Ao fazer isso, criamos qualidade na origem e geramos ou incorporamos conhecimento onde é necessário - isso nos permite criar sistemas de trabalho cada vez mais seguros, nos quais os problemas são encontrados e resolvidos muito antes de ocorrer uma falha catastrófica.

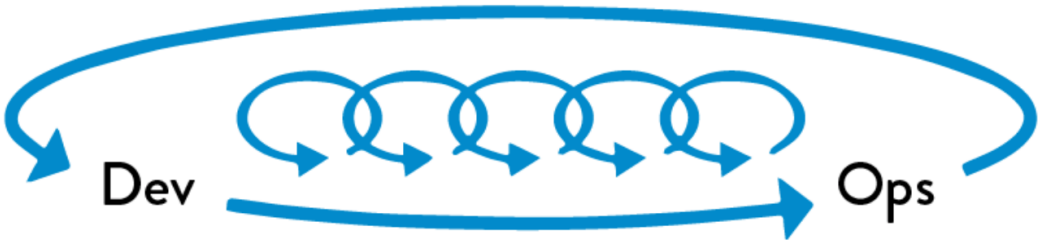


Figura 3: O terceiro caminho DevOps

O terceiro caminho possibilita a criação de uma cultura generativa de alta confiança que apoie uma abordagem dinâmica, disciplinada e científica à experimentação e tomada de riscos, facilitando a criação de aprendizado organizacional, tanto de nossos sucessos como fracassos. Além disso, ao encurtar e ampliar continuamente nossos ciclos de feedback, criamos sistemas de trabalho cada vez mais seguros e temos mais condições de assumir riscos e realizar experimentos que nos ajudam a aprender mais rápido do que nossos concorrentes e a ganhar no mercado.

1.4 Práticas DevOps

Embora não exista um corpo fechado de práticas técnicas DevOps, podemos propor uma lista de prática básicas para a implantação desta cultura. A lista aqui apresentada não possui uma ordem de prioridade, que depende da realidade de cada organização e da cultura já instalada nos seus times de desenvolvimento, qualidade e operações.

É importante que você, leitor, compreenda que as práticas são formas concretas de acionar os três caminhos DevOps descritos anteriormente.

As práticas DevOps buscam endereçar a melhoria nos atributos arquiteturais de qualidade interna descritos no começo deste capítulo, tais como a manutenibilidade, testabilidade ou implantabilidade. Com estas práticas implementadas, teremos maior garantia que a arquitetura definida pelo time de arquitetura será transformada em código executável e que minimizaremos o débito técnico dos produtos de software sendo construídos.

1.4.1 Comunicação Técnica Automatizada

A cultura DevOps busca aproximar pessoas dos times de desenvolvimento, qualidade e operações. E essa aproximação pode ser facilitada com ferramentas de comunicação que aliam o melhor da mensagem instantânea e canais de times com alarmes técnicos automatizados. Por exemplo, o Slack ou HipChat permitem que times com pessoas de desenvolvimento, qualidade e operações possam:

- estabelecer conversas texto, áudio e vídeo em canais privativos e focados;
- receber notificações automáticas tais como a disponibilização de builds, aprovação de releases ou problemas em produção;

- disparar comandos através de *bots* para a abertura de defeitos, escalonamento de builds ou releases.

1.4.2 Qualidade contínua do código

É fácil que uma arquitetura definida pelo time de arquitetura seja violada pelo time de desenvolvimento. Um arquiteto não tem tempo e dedicação para vigiar o código fonte e realizar a aderência arquitetural do código ou observar o uso de práticas apropriadas de codificação. O efeito é que a arquitetura executável colocada em produção pode ser diferente da arquitetura imaginada pelo arquiteto.

No sentido de minimizar esta lacuna, esta primeira prática lida com a automação da verificação da qualidade de código por ferramentas. Existem opções sólidas que permitem avaliar o uso das melhores práticas de programação no seu ambiente (*Code Metrics Tools*) e avaliar a aderência do seu código a uma arquitetura de referência (*Architectural Analysis Tools*). E essas podem ser programadas para rodar a noite ou até mesmo durante o momento do checkin do código pelo desenvolvedor. Existem ainda outras que facilitam o processo de revisão por pares dos códigos fontes (*Code Reviews*), estabelecendo workflows automatizados e trilhas de auditoria. O recurso de *Pull Requests* do Git é um exemplo deste mecanismo.

Com esta prática, temos robôs que atuam para facilitar a análise do código fonte, educar desenvolvedores e estabilizar ou mesmo reduzir o débito técnico instalado.

1.4.3 Configuração como Código

É comum que desenvolvedores façam muitas tarefas de forma manual. Exemplos incluem cópias de arquivos entre ambientes, configuração destes ambientes, configuração de senhas, geração de *release notes* ou a parametrização de aplicações. Esses trabalhos manuais são propensos a erros e podem consumir tempo valioso do seu time com tarefas braçais.

Times atentos devem observar quando algum tipo de trabalho manual e repetitivo começa a acontecer e buscar automatizar isso. A automação da configuração através da escrita de códigos de scripts é um instrumento DevOps importante para acelerar o trabalho de times de desenvolvimento, reduzir erros e garantir consistência do trabalho.

1.4.4 Gestão dos Builds

A gestão de builds (ou *Build Management*) é prática essencial para garantir que os executáveis da arquitetura sejam gerados de forma consistente, em base diária. Esta prática busca evitar o problema comum do código funcionar na máquina do desenvolvedor e ao mesmo tempo quebrar o ambiente de produção.

A automação de builds externaliza todas as dependências de bibliotecas e configurações feitas dentro de uma IDE para um script específico e que possa ser movida entre máquinas com consistência. Embora a automação de builds, na sua definição formal, lide apenas com a construção de um build, a prática comum de mercado é que builds devam executar um

conjunto mínimo de testes de unidade automatizados para estabelecerem confiabilidade mínima ao executável sendo produzido.

Esta prática lida ainda com o estabelecimento de repositórios confiáveis de bibliotecas, o que garante governança técnica sobre o conjunto de versões específicas de bibliotecas que estejam sendo usadas para montar uma aplicação.

1.4.5 Automação dos Testes

Organizações de baixa maturidade em automação de testes tem mais retrabalho ao longo de um projeto e geram mais incidentes em ambientes de produção. Esta prática DevOps buscar melhorar a testabilidade de aplicações e envolve:

- a criação de testes de unidade de código para as regras de negócio não triviais do sistema e também pontos críticos do código fonte tais como repositórios de dados complexos, controladores de negócio e interfaces com outros sistemas e empresas;
- a automação da interação com telas com testes funcionais automatizados;
- testes automatizados de aspectos não-funcionais, como segurança, acessibilidade, performance ou carga.

1.4.6 Testes de Carga

Em aplicações Web, podem haver variações abruptas no perfil de carga da aplicação em produção. Isso se deve a natureza estocástica no comportamento de requisições Web e a natureza do protocolo HTTP. Não é incomum que picos aconteçam e gerem 10x mais carga de trabalho que o uso médio da aplicação. Os sintomas comuns nas aplicações de mercado são longos tempos de resposta e até mesmo indisponibilidade do servidor Web (erros 5xx).

Estes sintomas podem ser minimizados com o uso de testes de carga em aplicações Web. O teste de carga é uma prática que permite gerar uma carga controlada para uma aplicação em um ambiente de testes específico ou homologação e assim estabelecer se um determinado build é robusto e pode ser promovido para ambientes de produção.

1.4.7 Gestão de Configuração

Times já usam ferramentas de controle de versão para organizar o seu código fonte, tais como o SVN, GIT ou Mercurial. Ao mesmo tempo, existem outras preocupações associadas ao trabalho feito por times em uma mesma linha de código. A ausência de políticas automatizadas de gestão de configuração digo aumenta a chance que desenvolvedores desestabilizem os troncos principais dos repositórios e gerem fricção e retrabalho para seus pares.

Neste contexto, existem opções de gestão de configuração de código permitem que os repositórios sejam mantidos em estado confiável e que erros comuns sejam evitados através da automação de políticas. Tarefas de gestão de configuração de código tais como a criação de rótulos (*labels*), geração de versões, mesclagem de troncos de desenvolvimento e a manutenção de repositórios podem ser aceleradas e tornadas mais consistentes com o uso

destes recursos. Como exemplo, o Git suporta o conceito de *hooks*[3], mecanismo extensível de automação de políticas de gestão de código. Outras como o GitLab, VSTS ou Atlassian Bamboo já possuem estes mecanismos embutidos.

1.4.8 Automação dos Releases

A automação dos releases (*Release Management*) é uma prática que buscar garantir que o processo de promoção do executável para os ambientes de testes, homologação e produção sejam automatizados e assim tornados consistentes. Isso é importante porque em muitas organizações é difícil colocar um produto em ambiente de produção. A demora para acesso aos ambientes, alto número de passos manuais, a complexidade e a dificuldade de analisar os impactos são comuns. Isso gera atritos, longas demoras e desgastes entre times de QA, desenvolvimento e operações. E no fim isso também provoca erros nos ambientes de produção.

Esta prática tem como principais benefícios:

- reduzir o tempo para entregar um novo *build* em ambiente de produção através da automação da instalação e configuração de ferramentas e componentes arquiteturais;
- reduzir erros em implantação causadas por parâmetros específicos que não foram configurados pelos times de desenvolvimento e operações;
- minimizar a fricção entre os times de desenvolvimento, QA e operações;
- prover confiabilidade e segurança no processo de implantar aplicações.

Um ponto de atenção é que a automação de releases não deve recompilar a aplicação. Para garantir consistência, ela deve garantir que o mesmo executável que foi montado na máquina do desenvolvedor (mesmo conjunto de bits) esteja operando em outros ambientes. Ou seja, a automação de releases faz a movimentação dos executáveis entre os ambientes e modifica apenas os parâmetros da aplicação e variáveis de ambiente. Este processo pode também envolver a montagem de máquinas virtuais em tempo de execução do script.

1.4.9 Automação da Monitoração de Aplicações

Ao colocarmos um build em ambiente de produção, devemos buscar que o mesmo não gere interrupções nos trabalhos dos nossos usuários. Muitas empresas ainda convivem com incidentes em ambientes de produção causados por falhas nos processos de entrega em produção e desconhecimento de potenciais problemas.

Uma forma de reduzir a chance de incidentes para os usuários finais é implementar a monitoração contínua de aplicações (*Application Performance Monitoring*). Esta prática permite que agentes sejam configurados para observar a aplicação em produção. Alarmes podem ser ativados para o time de operações se certas condições de uso forem alcançadas ou se erros inesperados surgirem. Isso permite ter ciência de eventuais incidentes com antecedência e tomar ações preventivas para restaurar o estado estável do ambiente de operações.

[3] https://git-scm.com/book/pt-br/v1/Customizando-o-Git-Um-exemplo-de-Pol%C3%ADtica-Git-Forçada

E, para melhorar a experiência, alguns times já fazem que estes alarmes sejam enviados através de *bots* para as ferramentas de comunicação tais como o HipChat ou Slack.

1.4.10 Testes de Estresse

O teste de estresse é um tipo de teste de carga onde queremos criar fraturas em uma aplicação dentro de um ambiente com parâmetros de hardware estabelecidos a priori. Ele consiste em aumentarmos a carga em uma aplicação para saber qual será o primeiro componente a falhar por sobrecarga (ex. Banco de dados, servidor Web ou fila de mensagem). Esta técnica permite que o time de operações possa priorizar a sua atenção em infraestruturas complexas. Ela também permite aos times de desenvolvimento estabelecer os limites de uso da sua aplicação para os seus clientes.

1.4.11 Integração Contínua (*Continuous Integration*)

Depois que a automação dos builds acontece, ela pode ser programada para ser executada em base diária ou até mesmos várias vezes por dia. Quando esta maturidade for alcançada, podemos avançar para que ela seja executada sempre, i.e., toda vez que um um *commit* acontecer em um código fonte.

É esperado que esta prática faça pelo menos o seguinte conjunto de passos:

- promova a recompilação do código fonte do projeto;
- execute as suítes de testes de unidade automatizados do projeto;
- gere o build do produto;
- crie um novo rótulo para o build;
- gere defeitos automatizados para o time se o build falhou por algum motivo.

A prática da integração continua promove as seguintes vantagens:

- detectar erros no momento que os mesmos acontecem;
- buscar um ambiente de gestão de configuração estável de forma continuada;
- estabelecer uma mudança cultural no paradigma de desenvolvimento, através de feedbacks contínuos para o time de desenvolvimento da estabilidade do build.

1.4.12 Entrega Contínua (*Continuous Delivery*)

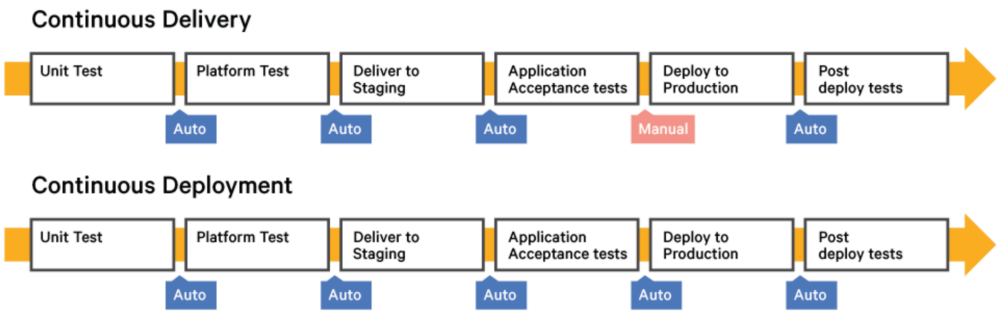
Em termos simples, a entrega continua é a automação do processo de promoção e publicação **nos diversos ambientes (testes, homologação ou produção)**. Ela é ativada por uma ferramenta automatizada quando um novo build for gerado.

Observe que a entrega contínua não significa que o ambiente de produção é modificado a todo instante. Apenas empresas de serviços de Internet podem e devem fazer isso. A entrega contínua implica que o ambiente de produção pode ser alterado se um novo build estiver disponível e as aprovações necessárias foram dadas para a promoção do build.

Na entrega contínua, o passo que antecede a publicação para a produção é realizado de forma manual.

1.4.13 Implantação Contínua (*Continuous Deployment*)

Na implantação contínua temos a automação de todo o ciclo de promoção, inclusive a cópia para o ambiente de produção. A figura disponível no sítio da Puppet traz a diferença entre esse processo e o processo de entrega contínua.



Fonte: puppet.com/blog/continuous-delivery-vs-continuous-deployment-what-s-diff

Integração, Implantação e Entrega Contínua – Um resumo

A *integração continua* (*Continuous Integration*) é o processo de compilar o código em ambiente limpo, rodar testes e outros processos de qualidade e gerar um build, disparado por qualquer modificação no código fonte.

A *entrega continua* (*Continuos Delivery*) é o processo de copiar o build gerado no processo de integração para ambientes QA, homologação ou produção. Na implantação contínua a etapa de promoção para produção é **manual**.

A *implantação continua* (*Continuos Deployment*) é o processo de copiar o build gerado no processo de integração para ambientes QA, homologação ou produção. Na implantação contínua a etapa de promoção para produção é **automatizada**.

1.4.14 Implantações Canário

As implantações canário são práticas úteis para empresas que praticam a entrega contínua e querem minimizar efeitos colaterais de novas funcionalidades na comunidade de usuários.

Quando um novo produto é colocado em produção pela automação dos releases, pode haver um risco de negócio em liberar as novas funcionalidades para toda a sua comunidade de

usuários. Talvez seja necessário fazer um certo experimento de negócio para saber se aquela funcionalidade será útil e mantida no produto.

Estes experimentos controlados podem ser ativados com os testes canários. O termo é devido a uma prática que ocorria nas minerações na Europa há alguns séculos. Mineiros levavam canários em gaiolas para novos veios em suas minas. Eles começam a trabalhar e deixavam os canários perto deles ao longo do dia de trabalho. Se um canário morresse depois de um tempo pequeno naquele ambiente, era porque o ambiente não estava saudável para a atividade humana devido a gases tóxicos. Na cultura DevOps da TI, a implantação canário consiste em habilitar novas funcionalidades apenas para um grupo controlado de usuários (os canários). Ou seja, em ambiente de produção a aplicação opera de duas formas distintas para duas comunidades (A/B). Isso pode ser implementado através do padrão de desenho chamado *Feature Toogle*s[4] e permite estabelecer uma prática chamada HDD (*Hypothesis Driven Development*). Se os canários não gostam do ambiente, a funcionalidade pode ser removida do produto em ambiente de produção sem intervenção no código fonte.

1.4.15 Infraestrutura como Código (*IAC*)

Times de baixa performance DevOps ainda possuem processos manuais e morosos de acionamento entre desenvolvimento e operações. Um exemplo prático é a criação de uma nova máquina feita do time de desenvolvimento para o time de operações. Em muitas empresas, este tipo de requisição demora horas, dias ou até mesmo semanas para ser realizada.

Quando times alcançam boa maturidade na configuração de elementos como scripts, elas podem avançar e tratar até o mesmo o hardware como código. Através de tecnologias disseminadas nos últimos anos em ambientes Linux, Windows ou OS/X, é possível configurar os processos de automação de build e automação de releases para criar uma máquina virtual através de um código de script. A infraestrutura como permite estabelecer para o time de operações confiabilidade apropriada para as novas implantações realizadas em ambientes de produção.

A infraestrutura como código traz ainda como grande benefício estabelecer um protocolo comum entre os times de desenvolvimento e o time de operações. Esta prática elimina a necessidade da criação manual de ambientes físicos, que é moroso, propenso a erros e que causa atrito entre times. Ao automatizar esse processo, podemos reduzir o tempo de ciclo para entrega de aplicações em produção. A tecnologia do Docker[5] é um excelente exemplo neste sentido.

Considere o exemplo do código a seguir. Nele vemos um arquivo em sintaxe Ansible, que é uma ferramenta de provisionamento de ambientes. Este arquivo define um estado de configuração desejado de um nodo de hardware em um ambiente. Ele verifica se o servidor Apache existe no cluster de máquinas denominado *webservers*. Se existir, ele baixa e instalar.

[4] http://martinfowler.com/articles/feature-toggles.html
[5] http://docker.com

Se já existir, ele avança o script. Após a instalação, ele verifica se o servidor está rodando. Se não estiver, ele sobe o servidor. Se ele já estiver rodando, ele não interfere no estado atual.

```
---
- hosts: webservers
  vars:
    http_port: 80
    max_clients: 200
  remote_user: root
  tasks:
  - name: ensure apache is at the latest version
    yum: name=httpd state=latest
  - name: write the apache config file
    template: src=/srv/httpd.j2 dest=/etc/httpd.conf
    notify:
    - restart apache
  - name: ensure apache is running (and enable it at boot)
    service: name=httpd state=started enabled=yes
  handlers:
    - name: restart apache
      service: name=httpd state=restarted
```

Figura 4: Arquivo de script Ansible
Fonte: http://docs.ansible.com

Através de utilitários como o Ansible, Power Shell DSC, Puppet, Chef, entre outras, estes arquivos de scripts podem ser executados em plataformas locais ou de nuvens como a Microsoft Azure ou Amazon EC2. A implantação deste arquivo cria uma máquina virtual com a exata especificação informada. Ou seja, todo o trabalho de criação manual de uma máquina virtual e sua tediosa configuração é eliminado. Ao invés, criamos e testamos um script em alguma linguagem e o ambiente subjacente se encarrega de fazer o provisionamento das máquinas virtuais no ambiente de produção.

1.4.16 Ambientes Self-Service

Em muitas empresas, é comum que um novato demore horas ou até dias para que consiga estabelecer um novo ambiente de trabalho. Isso é devido ao conjunto de passos manuais necessários, falta de procedimentos operacionais e dificuldades implícitas a montagem de ambientes LAMP, JS, Java EE ou .NET.

A prática de ambientes *self-service* permite que através de um código de script todo um ambiente de trabalho seja baixado, criado e disponibilizado para habilitar um desenvolvedor no seu trabalho em poucos minutos. Dado que um desenvolvedor tenha uma estação de trabalho com excelente memória RAM e uma rede veloz, é possível operacionalizar esta prática e salvar tempo precioso como o estabelecimento de ambientes de trabalho com confiabilidade e robustez. O Docker, em particular, é uma tecnologia que ganhou popularidade nos últimos anos para apoiar também esta prática.

1.4.17 Injeção de Falhas

Uma prática avançada DevOps é injetar defeitos no ambiente de produção de forma explícita. Por exemplo, podemos desligar o acesso ao banco de dados ou outros recursos críticos e forçar a aplicação a falhar. Isso pode parecer bizarro para alguns ou um contrassenso para outros. Mas pode fazer todo o sentido quando estamos buscando ambientes de alta disponibilidade e confiabilidade.

Através do uso de procedimentos controlados de desestabilização da aplicação, podemos verificar como a aplicação se recupera de uma falha em ambiente de produção. Algumas perguntas que o time de operações poderia investigar incluem:

- Ela se recupera e retorna para o estado original antes da falha?
- Ela emite alarmes apropriados para as partes interessadas?
- Ela fornece mensagens simples e explicativas para os usuários finais?

Esta prática começou a ganhar momento na TI depois que a Netflix publicou[6] o uso desta prática nos seus ambientes de produção e a disponibilização da sua ferramenta de injeção de falhas chamada *Simian Army*[7]. Ela permite estabelecer um mecanismo de antifragilidade[8] na sua aplicação, tornando-a melhor ao longo do tempo à medida que ela seja estressada.

1.4.18 Telemetria

A telemetria é uma forma avançada de monitoração de aplicações em ambiente de produção. Ela permite conhecer os padrões de uso de uma aplicação, variações de carga, acesso, entre outras questões. A Telemetria conta também com um mecanismo intrínseco de capacidade de análise de uso (*analytics*) que permite aos times conhecer os padrões de acesso e assim evoluir o produto do ponto de vista técnico e de negócio.

1.4.19 Planejamento de Capacidade

O planejamento de capacidade envolve o uso de técnicas estatísticas e teoria de filas para conhecer, modelar e simular a carga de trabalho em aplicações e assim estabelecer o hardware mais apropriado para rodar uma aplicação, bem como ter ciência dos limites e potenciais problemas de operação.

Esta técnica pode ser implementada por ferramentas de testes de carga e performance e são úteis para empresas que trabalhem com cenários desafiantes de cargas de trabalho e busquem o uso de computação elástica em ambientes de nuvens.

---

[6] https://www.infoq.com/br/news/2012/08/netflix-chaos-monkey
[7] https://github.com/Netflix/SimianArmy
[8] A fragilidade significa que algo quebra sob algum estresse, como por exemplo um copo fino solto de certa altura. Já a robustez indica que algo resiste a um estresse sem alterar o seu estado. Mas a anti-fragilidade vai além da robustez e resiliência. Um organismo anti-frágil melhora o seu estado depois de submetido a um estresse. Por exemplo, exercícios físicos, até um certo nível, geram estresses em pessoas e a resposta do corpo delas é se tornar melhor com uma melhor densidade óssea e maior massa muscular. A anti-fragilidade é o oposto matemático da fragilidade. Enquanto a fragilidade é denotada como um número negativo -X, a robustez seria denotada como o número 0 e a anti-fragilidade seria denotada como um número positivo X. Este conceito é apresentado e discutido no livro Anti-Frágil – Coisas que se beneficiam com o caos, de Nicholas Taleb, publicado em 2012.

Referências

Kim, G., Humble, J., Debois, P., & Willis, J. (2016). *The DevOps Handbook:: How to Create World-Class Agility, Reliability, and Security in Technology Organizations*. IT Revolution.

Kim, G., Behr, K., & Spafford, K. (2014). *The phoenix project: A novel about IT, DevOps, and helping your business win*. IT Revolution.

Humble, J., & Farley, D. (2010). *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation (Adobe Reader)*. Pearson Education.