Cap. 10 - DevOps (em andamento)

> *Imagine a world where product owners, development, QA, IT Operations, and Infosec work together, not only to help each other, but also to ensure that the overall organization succeeds.* – G. Kim, J. Humble, P. Debois, J. Willes

10.1 Introdução

Até agora, neste livro, estudamos um conjunto de práticas para desenvolvimento de software com qualidade e agilidade. Por meio de métodos ágeis — como Scrum, XP ou Kanban —, vimos que o cliente deve participar desde o primeiro dia da construção de um sistema. Também estudamos práticas importantes para produção de software com qualidade, como testes de unidade e refactoring. Estudamos ainda princípios e padrões de projeto e também padrões arquiteturais.

Logo, após aplicar o que vimos, o sistema — ou um incremento dele, resultante de um sprint — está pronto para entrar em produção. Essa tarefa é conhecida ainda pelos nomes de **implantação (deploy)** ou **entrega (release)** do sistema. Independentemente do nome, ela não é tão simples e rápida como pode parecer.

Historicamente, em organizações tradicionais, a área de Tecnologia da Informação costumava ser dividida em dois departamentos:

- Departamento de Sistemas (ou Desenvolvimento), formado por desenvolvedores, programadores, analistas, arquitetos, etc.
- Departamento de Suporte (ou Operações), no qual ficavam alocados os administradores de rede, administradores de bancos de dados, técnicos de suporte, etc.

Hoje em dia, é fácil imaginar os problemas causados por essa divisão. Na maioria das vezes, a área de suporte tomava conhecimento de um sistema na véspera da sua implantação. Consequentemente, a implantação poderia atrasar por meses, devido a uma variedade de problemas que não foram identificados antes. Dentre eles, podemos citar a falta de hardware para executar o novo sistema ou a nova funcionalidade, problemas de desempenho do novo sistema, incompatibilidades entre o banco de dados do novo sistema e o banco de dados de produção, vulnerabilidades de segurança, etc. No limite, esses problemas poderiam resultar no cancelamento da implantação e no abandono do sistema.

Resumindo, nesse modelo tradicional, existia um stakeholder importante — os administradores de sistemas ou *sysadmins* — que tomava conhecimento das características e requisitos não-funcionais de um novo software na véspera da implantação. Esse problema era agravado pelo fato de que os sistemas eram grandes monolitos, cuja implantação gerava todo tipo de preocupação, como mencionado no final do parágrafo anterior.

Para atenuar o problema, foi proposto então o conceito de **DevOps**. Por ser um termo recente, ele ainda não possui uma definição consolidada. Mas seus proponentes gostam de descrever DevOps como um movimento que visa unificar as culturas de desenvolvimento (Dev) e de operação (Ops), visando permitir a implantação mais rápida e ágil de um sistema. Esse objetivo está refletido na frase que abre esse capítulo, de autoria de Gene Kim, Jez Humble, Patrick Debois e John Willes, todos eles membros de um grupo de desenvolvedores que ajudou a difundir os princípios de DevOps. Segundo esses desenvolvedores, DevOps implica na seguinte disrupção na cultura tradicional de implantação de sistemas:

> Em vez de iniciar as implantações à meia-noite de sexta-feira e passar todo o fim de semana trabalhando para concluí-las, as implantações ocorrem em qualquer dia útil, quando todos estão na empresa e sem que os clientes percebam — exceto quando encontram novas funcionalidades e correções de bugs.

No entanto, DevOps não advoga a criação de um profissional novo, que fique responsável tanto pelo desenvolvimento como pela implantação de sistemas. Em vez disso, defende-se uma aproximação entre o pessoal de desenvolvimento e o pessoal de operações e vice-versa, visando fazer com que a implantação de sistemas seja mais ágil e menos traumática. Tentando explicar com outras palavras, a ideia é evitar dois silos independentes: desenvolvedores e operadores. Em vez disso, defende-se que esses dois profissionais conversem desde os primeiros sprints de um projeto. Para o cliente final, o benefício deve ser a entrada em produção mais cedo do sistema que ele contratou.

Para agilizar a implantação, os times ágeis podem incluir um profissional de operações, que participe dos trabalhos do time em tempo parcial ou mesmo em tempo integral. Sempre em função da demanda, o mesmo profissional pode também participar de mais de um time. A ideia é que ele antecipe problemas de desempenho, segurança, incompatibilidades com outros sistemas, etc. Ele pode também, enquanto o código está sendo implementado, começar a trabalhar nos scripts de instalação, administração e monitoramento do sistema em produção.

De forma não menos importante, DevOps advoga também a automatização de todos os passos necessários para colocar um sistema em produção e monitorar o seu correto funcionamento. Isso implica na adoção de práticas que já vimos neste capítulo, notadamente testes automatizados. Mas também de novas práticas e ferramentas, tais como Integração Contínua (*Continuous Integration*) e Entrega Contínua (*Continuous Deployment*), que iremos estudar neste capítulo.

Para finalizar, vamos discutir um conjunto de princípios para entrega de software, enunciados por Jez Humble e David Harley (link). Apesar de propostos antes da ideia de DevOps ganhar tração, eles são completamente alinhados com essa ideia. Alguns dos

princípios são os seguintes:

- **Crie um processo repetível e confiável para entrega de software**. Esse princípio pode ser considerado o mais importante deles. A ideia é que a entrega de software não pode ser um evento traumático, com passos manuais e sujeitos a surpresas. Em vez disso, colocar um software em produção deve ser tão simples como apertar um botão.
- **Automatize tudo que for possível**. Já comentamos sobre esse princípio antes nessa seção. Ele é um pré-requisito indispensável para atender ao princípio anterior. Advoga-se que todos os passos para entrega de um software devem ser automáticos, incluindo seu *build*, a execução dos testes, a configuração e ativação dos servidores e da rede, a carga do banco de dados, etc. De novo, idealmente, queremos apertar um botão e, em seguida, ver o sistema em produção.
- **Mantenha tudo em um sistema de controle de versões**. Por consequência, deve ser simples restaurar e voltar o sistema para um estado anterior. "Tudo" no enunciado do princípio refere-se não apenas a todo o código fonte, mas também arquivos e scripts de adminstração do sistema, documentação, páginas Web, arquivos de dados, etc.
- **Se um passo causa dor, execute-o com mais frequência e o quanto antes**. Esse princípio não tem uma inspiração masoquista. Em vez disso, a ideia é antecipar os problemas, antes que eles se acumulem e quando as soluções tendem a ser mais complicadas. O exemplo clássico é o de integração contínua. Se um desenvolvedor passa muito tempo trabalhando de forma isolada, depois ele — e também o seu time — podem ter uma grande dor de cabeça para integrar o código. Logo, como integração pode causar dor, a recomendação consiste então em integrar o novo código com mais frequência e o quanto antes, se possível, diariamente.
- **"Concluído" significa pronto para entrega**. Com frequência, desenvolvedores dizem que uma nova história está pronta (*done*). Porém, ao serem questionados se ela pode entrar em produção, começam a surgir "pequenas" pendências, tais como: a implementação ainda não foi testada com dados reais, ela ainda não foi documentada, ela ainda não foi integrada com o sistema X, etc. Esse princípio defende então que "concluído", em projetos de sofwtare, deve ter uma semântica clara, isto é: 100% pronto para entrar em produção.
- **Todos são responsáveis pela entrega do software**. Esse último princípio alinha-se perfeitamente com os princípios de DevOps que discutimos no início da seção. Ou seja, não admite-se mais que os times de desenvolvimento e operação trabalham em silos independentes e que seu comunicam apenas na véspera de uma implantação.

10.2 Integração Contínua

Servidores de Integração Contínua

10.3 Desenvolvimento no Trunk

Feature Flags

10.4 Entrega Contínua

10.5 Infraestrutura como Código

10.6 Engenharia de Releases