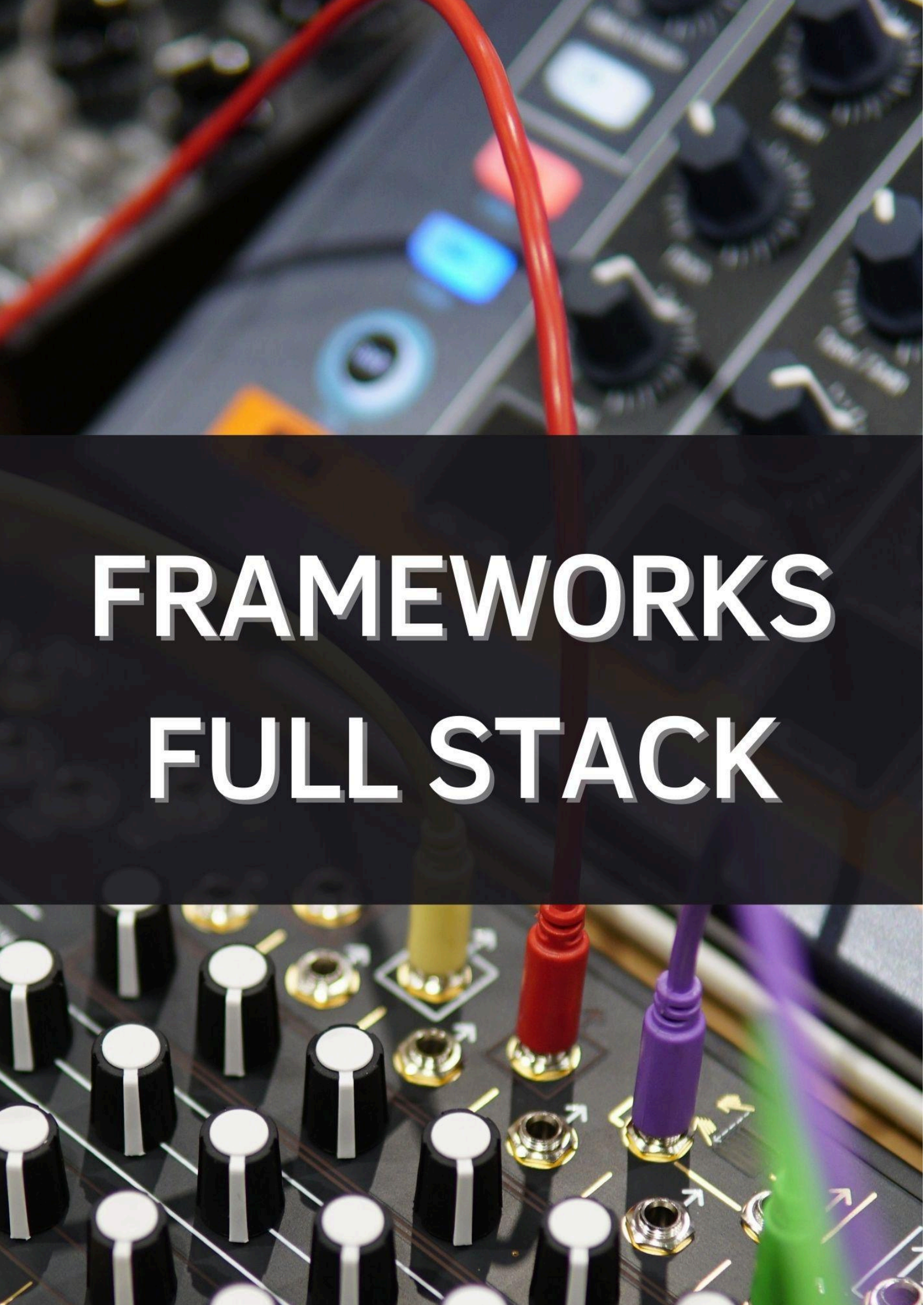

FRAMEWORKS
FULL STACK

Texto base

12

Continuous Delivery

Prof. André Nascimento Maia
Prof. Caio Nascimento Maia

Resumo

Uma introdução ao Continuous Delivery e seus benefícios, e instruções sobre como aplicar a prática de Continuous Delivery em duas Platform as a Service diferentes: Vercel para aplicações de front-end e a Python Anywhere para aplicações de back-end.

12.1. Introdução

A evolução e facilidade de entregas de ambientes complexos e completos de produção para grandes aplicações de *software* tem levado as empresas a entregar novas e melhores funcionalidades com maior frequência. Porém, como entregar com maior rapidez e segurança continuamente?

Uma das práticas que auxilia a empresa a aumentar sua rapidez e segurança nas entregas é o Continuous Delivery.

12.2. Continuous Delivery

Uma série de práticas foram desenvolvidas para garantir uma quantidade maior de entregas com rapidez e segurança de aplicações, principalmente no ambiente de nuvem. Estas práticas foram agrupadas e chamadas de Continuous Delivery (CD) (Humble, 2017). As práticas de CD guiam o desenvolvimento de software de forma sustentável e direciona a todo o time envolvido no projeto, desde de os engenheiros até os gerentes de projetos e gerentes de produtos, focados em melhoria contínua, sempre buscando melhorias diárias baseados nos princípios de agilidade e no movimento de Lean, buscando a identificação e eliminação de benefícios durante o desenvolvimento de um produto (Endeavor, 2015).

Através de fundamentos como gerenciamento de configuração, importante para a prática de Continuou Delivery, é possível guiar a arquitetura e planejar com maior precisão alterações em sistemas complexos e até de difícil acesso, como ocorre nos

casos de aplicações embarcadas. É pressuposto que as entregas em produção possam ser de quaisquer tipos, como: alteração de *software* para correção de *bugs* ou para adicionar novas funcionalidades; alterações de infraestrutura; alterações em bancos de dados; e qualquer outro tipo de configuração necessária para entregar aos usuários os benefícios de desenvolvimento de *software* contínuo.

Outros modelos que não focam tanto na entrega do resultado na mão do usuário ou em um ambiente de produção como o Continuous Delivery, geralmente criam estoques de funcionalidades, congelando o estado do código atual de produção e criando uma grande complexidade de sincronização do código em desenvolvimento e do que está em produção. A ideia por trás dos estoque de funcionalidades é permitir que times construam funcionalidades durante semanas ou meses e, somente quando tudo estiver pronto, então a funcionalidade completa é entregue em produção. A prática de Continuous Delivery pressupõem que o código esteja sempre em um estado pronto para ir para produção, permitindo entregas menores e mais frequentes.

Empresas têm utilizado esta prática para diminuir o risco nas entregas, atender as necessidades comerciais em tempo recorde, aumentar a qualidade do que está sendo desenvolvido, reduzir custos e sincronização entre ambientes e equipes, construírem melhores produtos testando e validando experimentos rapidamente e aumentando a felicidade das equipes de desenvolvimento (Humble, 2017).

12.2.1. Motivação

A grande motivação para uso do Continuous Delivery é criar uma situação do tipo ganha-ganha entre quem constrói os produtos e a empresa que vende o produto. Dentre vários benefícios, alguns são:

- **Baixo risco em entregas**. Elas são frequentes e automatizadas, então, elas se tornam algo prático e rotineiro. Além de incorporarem diversas estratégias de *rollout* como *Feature Flags* (Hodgson, 2017) e *rollback* como *blue-green deployment* (Fowler, Padegaonkar, 2010).
- ***Time to market* mais rápido**. Todo o processo de entrega de pequenas coisas é mais rápido, o que permite maior experimentação e velocidade nas respostas de mercado para todo tipo de negócio.
- **Maior qualidade**. Grande parte do processo é automatizado e os *feedbacks* são mais rápidos, o que permite entender e identificar problemas mais rapidamente e atuar sobre eles.

Outros benefícios como diminuição de custos, produtos melhores e mais aderentes às necessidades dos clientes, e até times de engenharias de software mais felizes porque a ocorre uma redução significativa na complexidade no processo de entrega de software estão relacionados a prática de Continuous Delivery (Humble, 2017).

12.2.2. Princípios

Continuous Delivery se baseia em 5 fundamentos que são independentes de tecnologia ou ferramental utilizados. São eles:

- **Qualidade embutida**. É necessário teste continuamente, principalmente de forma automatizada. Essa ideia de testes contínuos permite um *feedback* instantâneo sobre o estado de cada uma das funcionalidades. Geralmente, começamos com teste unitário que são mais simples e baratos de serem construídos, mas, conforme a complexidade aumenta são necessários testes mais complexos e abrangentes como teste de componentes, integração, até chegarmos em testes fim-a-fim e testes exploratórios.
- **Trabalhar em pequenos lotes**. Entregar funcionalidades completas e muito grandes deve ser evitado quando se utiliza Continuous Delivery. O pensamento deve ser de construir uma funcionalidade pouco a pouco, como um pintor faz para pintar um quadro: camada sobre camada, até que a pintura esteja de seu agrado e nítida o suficiente. Frequentemente, utilizando o processo de pequenas entregas combinado com a validação e *feedbacks* contínuos, uma funcionalidade pode mudar de direção para se encaixar as novas necessidades de clientes que não foram consideradas desde o início.
- **Computadores executam tarefas repetitivas, pessoas resolvem problemas**. Em Continuous Delivery nada de fazer a mesma coisa repetidamente. Todas as tarefas que podem ser automatizadas, devem ser automatizadas e deixar que o computador, a máquina, faça o que faz de melhor e com a precisão que é capaz. As pessoas devem se concentrar em resolver problemas complexos e automatizá-los. Esse princípio garante um nível adicional de qualidade, porque as pessoas são péssimas em fazer tarefas repetitivas porque sempre existe a chance de um erro humano.
- **Busque incansavelmente a melhoria contínua**. Esse princípio está intimamente ligado aos princípios de desenvolvimento de software ágil. Não há nada de bom que não possa melhorar. A empresa ou time que abraça a prática de Continuous Delivery deve estar preparada para modificações constantes, elas devem fazer parte do processo e cultura das pessoas envolvidas. Transformar Continuous Delivery em um projeto que precisar ser aplicado durante um período, não respeita esse princípio.
- **Todos são responsáveis**. Todos em um time devem ser responsáveis pelo o que está sendo construído. Não existe o problema de Fulano ou Sicrano, o problema é sempre de todos. O *feedback* rápido sobre a opinião dos clientes que estão experimentando uma funcionalidade ou melhoria que acabou de ser entregue em produção é uma ferramenta poderosa para colocar o time como responsável de tudo o que está sendo construído.

12.2.3. Fundamentos

Para atingir os princípios de Continuous Delivery, a prática se baseia em 3 fundamentos: Gestão de configuração, Continuous Integration e Teste Contínuo.

A gestão de configuração permite uma mentalidade e prática de automatizar todo o processo de entregas de qualquer alteração em qualquer ambiente através de *scripts* ou ferramentas e serviços que possibilitem a abstração adequada de determinada configuração. Como de configuração, podemos citar: recursos alocados em nuvem como filas e máquinas virtuais; parâmetros de configuração de determinados serviços; lançamentos de bancos de dados; configurações de bancos de dados. Toda configuração

deve ser versionada para permitir entendimento de configuração anteriores e análise das mudanças ao longo do tempo.

O gerenciamento de configuração tem dois objetivos principais: reprodutibilidade, que permite o provisionamento de ambientes complexos a qualquer momento, bastando apenas a execução dos *scripts*; e rastreabilidade, que permite comparar ambientes e entender, por exemplo, as versões de uma biblioteca utilizada em diferentes ambientes ou configurações específicas para cada um deles.

Benefícios como recuperação de desastres, ambiente auditável, gestão de capacidade e rápida resposta aos bugs encontrados são atingidos com o gerenciamento de configuração.

Continuous Integration é outro fundamento de Continuous Delivery. É preciso combinar e sincronizar o trabalho de diversos engenheiros e engenheiras de software enviando código continuamente para produção. Continuous Integration permite um *feedback* rápido sobre qualquer problema de forma padronizada, garantindo que o código sempre esteja pronto para ser entregue em produção, acompanhando um dos princípios desta prática.

O fundamento de Teste Contínuo é necessário para garantir o *feedback* preciso sobre toda e qualquer etapa do processo. Testar continuamente deve fazer parte do processo de entrega de todas as alterações em produção, seja para uma configuração de ambiente em nuvem, para uma alteração na estrutura de tabelas de uma banco de dados ou para uma alteração de código, é necessário garantir que uma simples e pequena alteração não afete significativamente a experiência do cliente em produção.

12.3. Exemplo de Continuous Delivery com PaaS

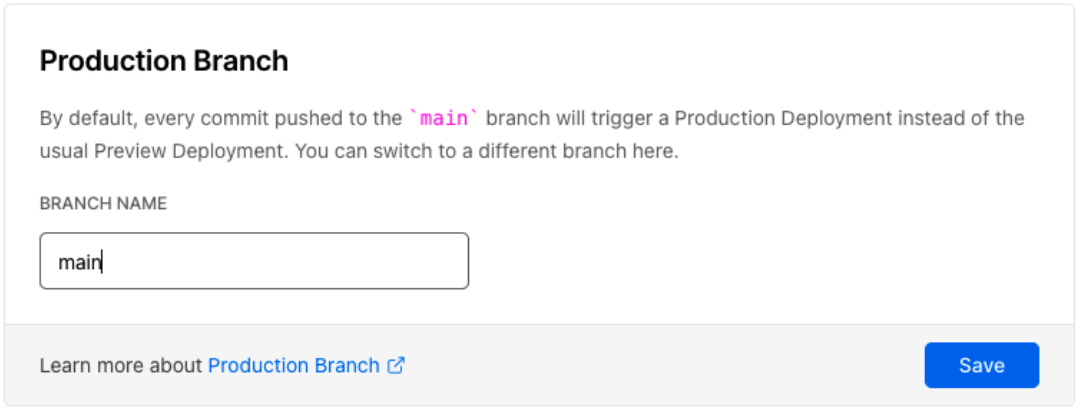
Para exemplificar um processo simples de CD, comentaremos sobre a Vercel e o Python Anywhere que têm esse processo embutido em suas plataformas.

Ambas as plataformas disparam o processo de entrega contínua em produção através de commits em *branches* específicas. Basta apenas configurar qual a *branch* que, toda vez que essa *branch* sofrer um *commit,* o processo de entrega contínua é iniciado e a versão atual em produção é substituída pelo novo pacote ou versão de aplicação.

A Vercel e a Python Anywhere são plataformas utilizadas como serviços, então elas abstraem grande parte do trabalho de implementação das ferramentas geralmente utilizadas na prática de Continuous Delivery. Todo o *pipeline* de entrega e gerenciamento de configuração acontecem de forma transparente, bastando apenas algumas configurações básicas e *commits* em *branches* específicas de um repositório Git.

Na Vercel, a configuração padrão da *branch* de entrega do repositório é a *main*. No caso da Vercel, existe na plataforma uma verificação contínua das alterações de um repositório do Github. A Figura 12.2 exibe a configuração de *branch* da Vercel.

Figura 12.2: Configuração de branch de deployment



Fonte: autores, 2022.

Referências

Endeavor. **Lean startup:** o que é e como aplicar. Endeavor Brasil, 07 out. 2015. Disponível em: <https://endeavor.org.br/estrategia-e-gestao/lean-startup/>. Acesso em: 18 fev. 2022.

FOWLER, Martin. **BlueGreenDeployment.** Ilustração de Ketan Padegaonkar. Martin Fowler, 01 mar. 2010. Disponível em: <https://martinfowler.com/bliki/BlueGreenDeployment.html>. Acesso em: 18 fev. 2022.

Hodgson, P. **Feature Toggles (aka Feature Flags)**. Martin Fowler, 2017. Disponível em: <https://martinfowler.com/articles/feature-toggles.html>. Acesso em: 18 fev. 2022.

Humble, J. **What is Continuous Delivery?** - Continuous Delivery, 2017. Disponível em: <https://continuousdelivery.com/>. Acesso em: 09 fev. 2022.