DevOps

UNIDADE 07

A importância de testar as coisas

Fazendo as Coisas Direito: Incluindo Testes

Quero mostrar para você a importância de testes com um exemplo da vida real. Vamos supor que você tem o sonho de comprar um carro 0 km e, após algum tempo trabalhando com TI, você conseguiu juntar dinheiro o suficiente para isso. Legal, né? Agora, pense que esse carro passou por uma linha de montagem em uma fábrica de automóveis. Cada carro passa por várias etapas: a montagem do chassi, a instalação do motor, a sua pintura, a instalação do acabamento, entre outros passos. E esse carro chegou na sua mão. Contudo, ninguém na fábrica testou o seu carro, e você recebeu um veículo defeituoso no final das contas. Mesmo que seja 0 km, as portas não fecham direito, os freios possuem problemas e a pintura está descascando. Imagino que você não ficaria feliz com isso, certo?

Agora pense que, em vez de uma linha de montagem de carros, temos um *pipeline* CI/CD para desenvolvimento de *software*. Ainda que tenhamos feito os passos de CI e CD na semana 3, não falamos muito sobre os testes, não é? Sem incluir testes no processo, você corre o risco de lançar um produto cheio de problemas para os usuários. Vamos entender juntos o que são os **testes unitários** e **testes integrados**?

Implementação de Testes

A ideia de implementarmos testes em nossos trabalhos de DevOps implica garantirmos um ciclo de desenvolvimento mais seguro. Lembra quando falamos das métricas anteriormente? Pois bem: para reduzirmos as chances de termos uma implantação (*deploy*) com erros, é necessário nos prevenirmos. E essa prevenção é feita com a prática dos testes.

Pipelines de CI/CD com testes

No contexto de DevOps, a intenção é implementarmos **testes automatizados** na *pipeline* de CI/CD. Cada vez que alguém faz uma alteração no código e envia uma PR (*pull request*), queremos ter certeza de que essa mudança não vai quebrar nada. Nos exemplos em que trabalhamos até o momento, isso significa, na prática, customizarmos o GitHub Actions para termos uma ação que, sempre que uma PR é enviada, executa os nossos testes automaticamente.

Por exemplo, podemos ter um arquivo de configuração YAML no GitHub que diz ao GitHub Actions para rodar os nossos **testes unitários** primeiro. Se todos os testes unitários passarem, ele pode então rodar os **testes de integração** (ou testes integrados). Se passar, estaríamos prontos para o *merge* da PR.

A implementação de testes em um *pipeline* de CI/CD começa com os **testes unitários**. Esses testes verificam se pequenas partes do código, como funções ou métodos individuais, funcionam corretamente. Se me permite a analogia, é como se testássemos somente o volante, ou somente os pedais, ou somente o motor de um carro **separada e individualmente**. Agora, pensando em casos reais: se estamos desenvolvendo um aplicativo de *e-commerce*, os testes unitários podem verificar se a função que calcula o preço total de um carrinho de compras está funcionando corretamente, ou se a função de finalizar a compra está funcionando como deveria. Se houve algum erro, paramos por aqui e informamos ao desenvolvedor dono da PR para que resolva as falhas nos testes. Dá para usarmos ferramentas como JUnit (para Java), NUnit (para .NET), PyTest (para Python) e Jest (para JavaScript) para isso.

Agora, e se deu certo? Aí avançaríamos para os **testes integrados** (ou testes de integração). Eles verificam se as diferentes partes do sistema funcionam bem **juntas**. Voltando ao nosso exemplo do *e-commerce*, isso poderia incluir alguns testes que verificam se o processo de finalização do carrinho funciona corretamente, integrando o cálculo de preços, o processamento de pagamentos e a atualização do estoque. Esses testes integrados são

mais complexos e demoram mais para serem executados, mas mesmo assim são essenciais para garantir que o sistema como um todo funcione sem problemas. Para isso, poderíamos usar ferramentas como o Selenium (múltiplas linguagens), Cypress (JavaScript), Playwright (múltiplas linguagens), Cucumber (múltiplas linguagens) e Puppeteer (múltiplas linguagens).

Também podemos configurar essas ferramentas em conjunto com o GitHub Actions. Vamos testar juntos?

DevOps x DevSecOps

Agora que você já está familiarizado(a) com todo o processo de DevOps, é hora de darmos um passo adiante e introduzirmos o conceito de **DevSecOps**. DevSecOps é a integração de práticas de segurança ao longo do ciclo de vida de desenvolvimento e operações.

DICA

Se DevOps é *development* + *operations*, DevSecOps é *development* + ***security*** (segurança) + *operations*. Logo, é DevOps + processos de cibersegurança.

Enquanto o DevOps foca a colaboração entre desenvolvimento e operações para entregar *software* mais rápido e com maior qualidade, o DevSecOps adiciona uma camada extra para garantir que a segurança seja uma prioridade desde o início.

Pense no DevSecOps como uma extensão natural do DevOps. No desenvolvimento tradicional, a segurança muitas vezes é vista como um passo final, algo a ser verificado depois que o código já está pronto e prestes a ser lançado. Isso pode levar a descobertas por vezes atrasadas de vulnerabilidades. E descobrir uma vulnerabilidade tarde demais pode implicar em uma complexidade e custo adicionais para arrumar – sem contar nos riscos de segurança.

Usar DevSecOps significa, na prática, usar ferramentas e práticas como **análise estática de código**, **testes de segurança automatizados e monitoramento contínuo**. Por exemplo, podemos configurar ferramentas de análise estática para verificar nosso código em busca de vulnerabilidades conhecidas a cada *commit*. Ou,

quem sabe, automatizar testes de segurança para rodarem junto com nossos testes unitários e integrados. Veja a comparação entre DevOps e DevSecOps na tabela abaixo – de fato, um dos aspectos principais dessa mudança está na inclusão de testes de segurança ao longo do processo.

Característica	DevOps	DevSecOps
Integração contínua (CI)	Testes automatizados básicos durante a integração de código.	Testes automatizados mais avançados, incluindo testes de segurança e vulnerabilidades.
Entrega contínua (CD)	Foco em testes funcionais e de regressão antes do *deploy*.	Testes adicionais para verificar a segurança, como análise de código estático e dinâmico.
Segurança	Segurança tratada como um passo separado, geralmente antes do *deploy*.	Segurança integrada no *pipeline* de CI/CD, com testes de segurança contínuos e automatizados.
Automação de testes	Automação básica para testes unitários e integração.	Automação abrangente, incluindo testes de segurança, análise de dependências e monitoramento contínuo.

Assim, a ideia do DevSecOps é a de incorporar práticas de segurança desde o início do desenvolvimento, automatizando verificações de segurança e integrando essas práticas nos *pipelines* de CI/CD. Assim, cada mudança no código é avaliada não apenas pela funcionalidade e qualidade, mas também pela segurança, reduzindo o risco daquelas vulnerabilidades.

REFLEXÃO

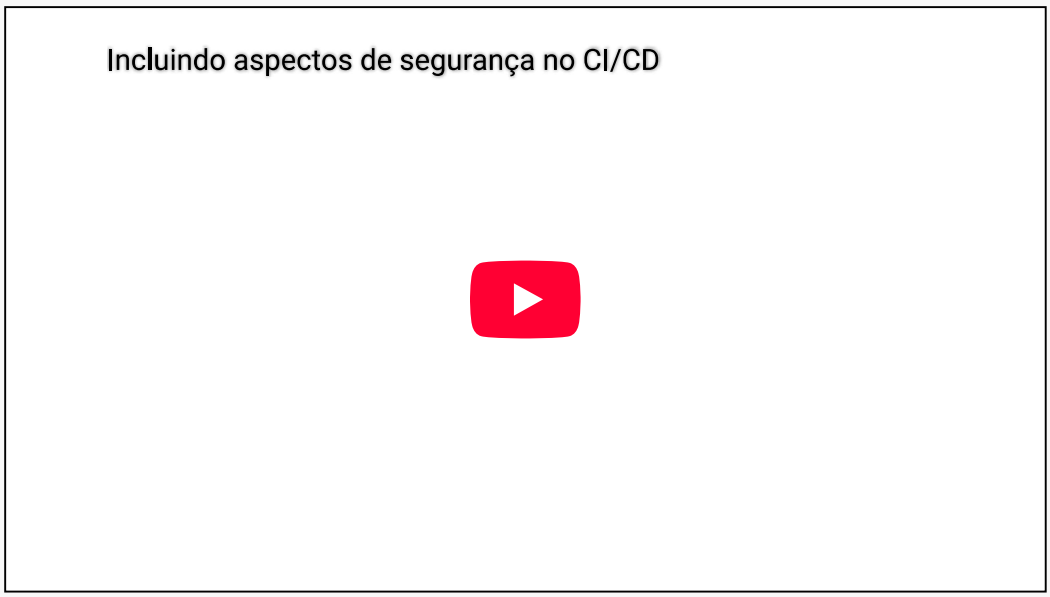
Você pode se perguntar: se é tão importante assim, por que é mais comum falarmos de DevOps do que DevSecOps?

Se nem todas as empresas conseguem um alto grau de maturidade em DevOps, imagine a dificuldade para implementarmos DevSecOps em todos os lugares. De fato, usar DevSecOps pode ser desafiador para algumas empresas, porque isso significa **mais complexidade e um custo maior**.

Explicando melhor: integrar práticas de segurança em todo o ciclo de vida do desenvolvimento exige ferramentas especializadas e também treinamento adequado para os desenvolvedores e operações. Muitas empresas já possuem processos DevOps estabelecidos e a transição para DevSecOps pode significar uma reestruturação significativa de todos os processos. Para piorar, esses custos iniciais podem ser proibitivos para pequenas empresas ou *startups*, levando-as a priorizar a implementação básica de DevOps antes de adicionar camadas adicionais de segurança.

Além disso, DevSecOps não resolve todos os problemas de segurança. Enquanto integra segurança ao processo de desenvolvimento, não elimina a necessidade de outras práticas de segurança, como auditorias regulares e testes de penetração (também chamados de *pentests*). Algumas empresas também podem achar

que o seu nível atual de segurança é adequado para seus riscos e requisitos específicos e, portanto, não veem uma necessidade urgente em adotar DevSecOps.



Conclusão

Nesta semana, conversamos sobre testes em DevOps, além do conceito de DevSecOps. Começamos com os testes unitários e integrados, que são essenciais para verificar a funcionalidade e a interação dos diferentes componentes do sistema. Esses testes, automatizados e executados regularmente, asseguram que o código esteja funcionando conforme o esperado e que problemas sejam identificados rapidamente.

Além disso, introduzimos o conceito de DevSecOps, que transforma a segurança em uma parte integrante do ciclo de desenvolvimento, e não apenas uma etapa final. Com isso, estamos quase perto da conclusão da nossa disciplina. Vamos para mais uma semana?

Referências

FREEMAN, E. **DevOps para leigos**. Rio de Janeiro: Editora Alta Books, 2021. *E-book*.

KIM, G.; BEHR, K.; SPAFFORD, G. **O projeto Fênix**. Rio de Janeiro: Editora Alta Books, 2020. *E-book*.

KIM, G.; HUMBLE, J.; DEBOIS, P.; WILLIS, J. **Manual de DevOps**. Rio de Janeiro: Editora Alta Books, 2018. *E-book*.