**DevOps – Quality Assurance - QA**

Nesse nosso DoJo vamos abordar a Garantia da Qualidade – QA(Quality Assurance) no DevOps. Como gostamos muitos de conceito, o que é DevOps? Uma **cultura** composta por **boas práticas auxiliada por ferramentas** que permitem **entregar softwares muito mais rápidos e com muito menos custos** do que comparada a metodologias tradicionais de desenvolvimento de software e gerenciamento de infraestrutura.

**Porque utilizar DevOps?**

Em uma empresa sem **DevOps** o lançamento de aplicações são atividades estressantes e de alto risco, que envolvem várias equipes diferentes. Por isso são conhecidos com **a última milha** tamanho são os esforços e as dificuldades encontradas. Com o movimento Ágil foi cunhado o termo **DevOps**para a **integração entre as equipes de desenvolvimento, qualidade e infraestrutura.**

Como DevOps é um tema muito extenso para um único DoJo vamos focar nossos esforços no Quality Assurance – QA (Garantia da Qualidade).

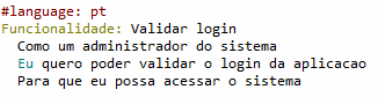
Vamos utilizar a temática de questionamentos para o desenvolvimento de nossas atividades, o intuito não é esgotar o nosso assunto, mas sim iniciar um fórum de discurssão.

Quais são os testes que podemos efetuar para que um código seja integrado em ambiente de produção com qualidade? Frameworks e Ferramentas.

**BDD**

Em nossa abordagem vamos focar nossos estudos em práticas **DevOps** é o primeiro será o **BDD (Behavior-Driven Design/Development ou Design/Desenvolvimento guiado por comportamento)**, onde é uma técnica voltada para o comportamento da aplicação e pessoas não técnicas compreendem os testes e podem escrever-los também.

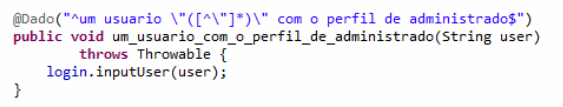
Observe um código BDD.



Principal intuito do BDD é manter uma linguagem única com todos os envolvidos no projeto (Stakeholders), ou seja, qualquer pessoa seja ele conhecedora de técnicas de programação ou não consegue entender o código.

No nosso exemplo estamos utilizando o Cucumber por ser o framework mais conhecido para testes de comportamentos e implementação em várias linguagens de programação.

Após a criação do BDD é gerado o método que efetuará o comportamento.



Após o desenvolvimento o Cucumber, teste automatizado de aceitação, é executado para uma aceitação automatizada da funcionalidade, gerando principalmente relatórios e capturas de telas como evidência.

TDD

Prosseguindo na etapa de desenvolvimento da funcionalidade podemos fazer uso de outros técnicas, antes de implementar a funcionalidade podemos escrever testes para guiar o nosso desenvolvimento os TDD´s - Test Driven Development .

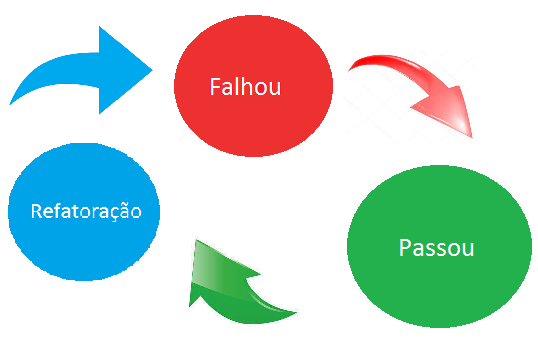


Figura xx - Ciclo de desenvolvimento guiado por testes

TDD é Baseado em pequenos ciclos de repetição, onde para cada funcionalidade do sistema um teste é criado antes, inicialmente falhando, por não existir a implementação, posteriormente criamos a funcionalidade e o teste passa e efetuamos um melhoramento do código com uma refatoração.

Testes Unitários

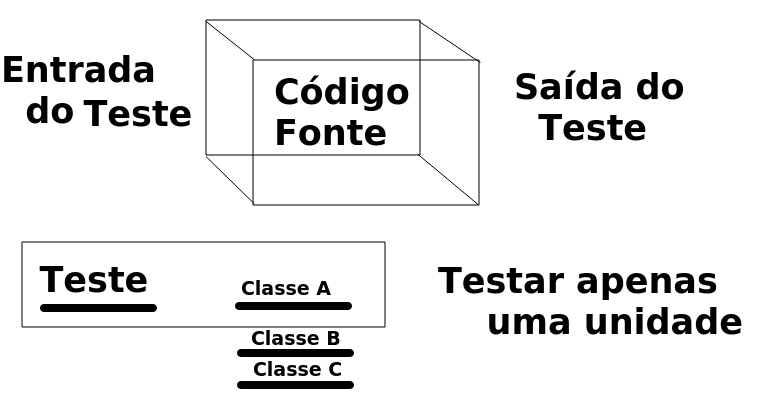


Figura xx - Estrutura de testes de unidade

Testa uma única unidade do sistema, na orientação a objetos, uma classe por exemplo, de maneira isolada e automatizada.

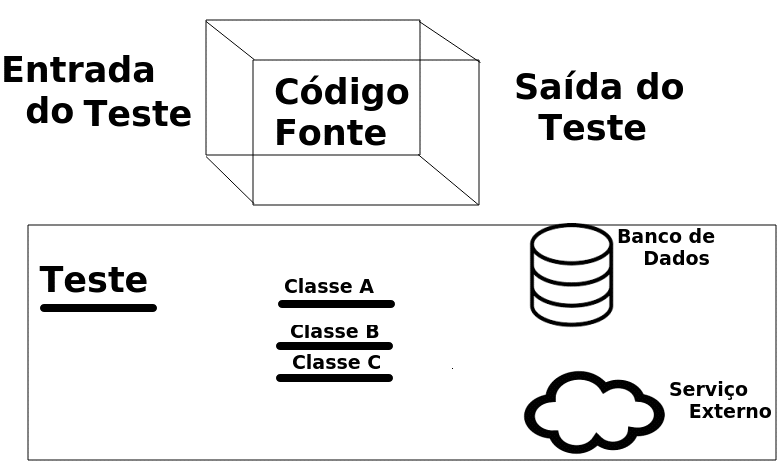


Figura XX - Estrutura de testes automatizados

Testa a integração entre dois ou mais módulos do sistema; na orientação a objetos, uma classe de serviço que efetua uma consulta na base de dados, por exemplo; de maneira completa e automatizada.

Testes Exploratórios

Sim. Mesmo com BDD, TDD, testes unitários e de testes de integração são necessários os testes exploratórios.

Para você que não sabe o que é um Teste Exploratório, definição, é um estilo de teste no qual você explora o software enquanto, simultaneamente, escreve e executa os testes usando o feedback do último teste para executar o próximo.

Teste de Performance

Aquele em que submetemos o sistema a uma avaliação de carga, stress ou desempenho, avaliando os resultados para comparar com as especificações inicias.

Podemos citar o JMeter que consegue efetuá-lo.

Figura XX - Executando testes com o JMeter

Figura XX - Relatório JMeter

Como é possível simular o ambiente de produção no ambiente de testes?

Inicialmente era muito difícil simular o ambiente de produção em nossos testes, atualmente, configurações de bibliotecas, memória, Hadware eram distintas. Com o advento da virtualização é possível ter ambientes similares.

Além das soluções de mercado amplamente conhecidas: Oracle Virtualbox e VMware. Temos o Docker uma tecnologia que nós oferece Containers, sendo uma camada entre o sistema operacional e nossas aplicações, permitindo a portabilidade para outro Docker que está em outro Sistema Operacional por exemplo.

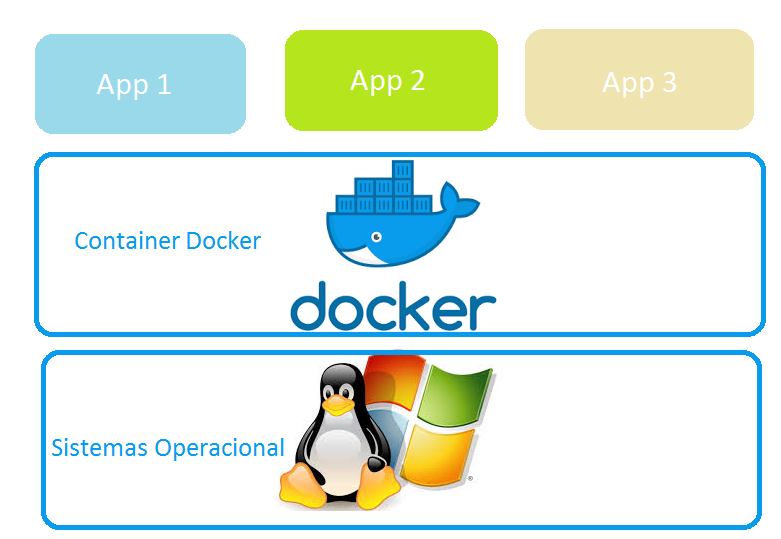


Figura XX - Estrutura Container Docker

Bom, agora vamos entender como automatizar o processo de entrega com Pipeline e como o Jenkins pode auxiliar na automação de testes?

Antes mesmo da cultura DevOps já trabalhávamos com entrega contínua e execução dos diferentes tipos de teste executados pelos Jenkins, todavia, precisamos explorar ao máximo esses recursos, primeiro vamos abordar a questão de Pipeline que são processos automatizados para cada etapa do nosso do projeto.

Por exemplo, no ambiente de desenvolvimento queremos que sejam executados a build do projeto, a execução dos testes unitários e de integração, uma avaliação da qualidade do código com o Sonar e a publicação no ambiente de desenvolvimento. Já no ambiente de homologação não vamos nos atentar aos testes unitários e de integração, mas sim, queremos executar os testes de aceitação automatizados e os testes de performance. Já no ambiente de produção queremos que sejam executados apenas os testes de canários(Testes A/B) que são testes de experimento das funcionalidades ou também conhecidos como testes de fumaça. Cada um desses processos supra citados podem ser agrupado em uma Pipeline para cada ambiente, invés de serem executados separados e controlados manualmentes.

Figura XX - Pipeline de Desenvolvimento

Figura XX - Pipeline de Homologação

Figura XX - Pipeline de Produção

O Jenkins também possui inúmeros plugins para auxiliar na qualidade, separamos alguns relevantes: JunitPlugin (Tendência de Resultados de Testes), Test Results Analyser(Consolida e apresenta o resultado de todos os testes), Cucumber Living Documentation(Gera documentação completa em PDF e HTML baseado nas User Stories do Cucumber).

Por fim vamos falar um pouco de Git Flow - Como utilizar o Git Flow na Garantia da Qualidade?

Como amamos definições Git Flow é modelo de organização de branch. Simples! Estabelece regras de nomenclatura para cada tipo de branch.

Resumidamente temos as branch Master que contém o código de produção, develop que contém o código para o próximo deploy em produção e serve de base para novas funcionalidades e sempre é atualizada por uma feature, falando em feature são as branch onde novos recursos são desenvolvidos, ainda temos as hotfix que são utilizadas para corrigir o código de produção e encerrada atualizada tanto a branch master como a branch develop, temos também as releases que são marcos extraídos da develop antes da incorporação com a master.



**Figura XX - Repositório Git organizado com Git Flow**

Git Flow permite que uma funcionalidade seja testada isoladamente na sua feature pela equipe de qualidade, sem afetar o andamento dos demais testes nas demais branch. Outra ponto interessante é que é possível testar uma correção apontando o código para uma hotfix. Grande ganho de se utilizar o Git Flow na garantia da qualidade é que podemos efetuar a liberação de partes menores e maior controle de versão.