

# 4Linux também é consultoria, suporte e desenvolvimento.

- Você já deve nos conhecer pelos melhores cursos de linux do Brasil mas a 4Linux também é **consultoria, suporte e desenvolvimento** e executou alguns dos mais famosos projetos do Brasil utilizando tecnologias “open software”.
- Você sabia que quando um cidadão faz uma aposta nas loterias, saca dinheiro em um ATM (caixa eletrônico), recebe um SMS com o saldo de seu FGTS ou simula o valor de um financiamento imobiliário no “feirão” da casa própria, ele está usando uma infraestrutura baseada em softwares livres com consultoria e suporte da 4Linux?

## → Serviços

### Consultoria:

Definição de Arquitetura, tuning em banco de dados , práticas DEVOPS, metodologias de ensino on-line, soluções open source ( e-mail, monitoramento, servidor JEE), alocação de especialistas em momentos de crise e auditoria de segurança.

### Suporte Linux e Open Source:

Contratos com regimes de atendimento – preventivo e/ou corretivo - em horário comercial (8x5) ou de permanente sobre-aviso (24x7) para ambientes de missão crítica. Suporte emergencial para ambientes construídos por terceiros.

### Desenvolvimento de Software:

Customização visual e funcional de softwares Open Source, consultoria para a construção de ambiente ágeis de Integração Contínua (Java e PHP) e mentoria para uso de bibliotecas, plataformas e ambientes Open Source. Parceiro Oficial Zend.

## → Parceiros Estratégicos



## Saiba mais

(11) 2125-4769  
[www.4linux.com.br/suporte](http://www.4linux.com.br/suporte)  
[contato@4linux.com.br](mailto:contato@4linux.com.br)



# Integração e Entrega Contínua com GIT, Jenkins, Nexus e Sonar

Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

# 4LINUX / Sumário

2



DevOps página 3



Git página 24



Gitlab página 85



Jenkins

Jenkins página 124



Sonarlint página 181



Sonatype Nexus página 204



Selenium página 231

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

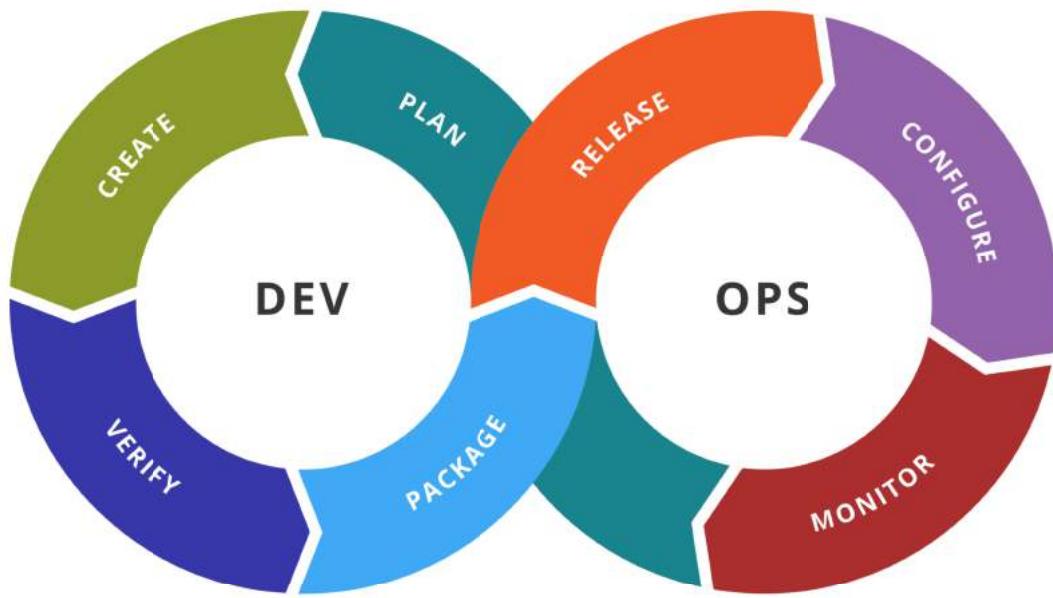
---

---

---

---

---



## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---



**DevOps** originou-se a partir da junção das palavras: ‘Desenvolvimento’ e ‘Operações’. Trata-se de um conjunto de **melhores práticas** que enfatizam a **colaboração** e a **comunicação** de profissionais de TI, desenvolvedores, operadores e pessoal de suporte, no **ciclo de vida de aplicativos e serviços**.



### Conceito DevOps

DevOps visa estabelecer uma cultura e ambiente onde a construção, testes e liberação de software e serviços possam acontecer rapidamente, com frequência e com suporte mínimo, a fim, de aumentar continuamente a vantagem do negócio e a satisfação do cliente.

### Os principais elementos do DevOps

Implementar uma cultura diversa com base na confiança, para identificar falhas rápidas e frequentes.

Fornecimento de aplicativos e serviços para negócio Just-in-Time (JiT), agilizando a entrega contínua (Agile) e assegurando feedback frequentes.

Garantir a continuidade dos serviços de TI através de uma abordagem de risco, baseada nas necessidades do negócio.

Gerenciamento do ciclo de vida de aplicativos e serviços, incluindo condições de ciclo vida.



**DevOps** enfatiza a cooperação por toda a organização, a automação de entregas e mudanças na infraestrutura, conduzindo à **Continuous Integration, Continuous Deployment** e **Continuous Delivery**.



### DevOps

Muda o modo de pensar dos indivíduos em relação ao seu trabalho.

Valoriza a diversidade do trabalho realizado, apoia processos intencionais que aceleram a taxa por meio da qual as empresas percebem valor e, mede o efeito das mudanças nos âmbitos social e técnico.

Constitui uma forma de pensar, permite que indivíduos e organizações desenvolvam e mantenham práticas de trabalho sustentáveis.

## Princípios DevOps

- ✓ **Integração contínua:** fácil transferência de controle do desenvolvimento para operações e suporte.
- ✓ **Implantação contínua:** liberação contínua ou tão frequente quanto possível.
- ✓ **Feedback contínuo:** buscar feedback das partes interessadas durante todas as fases do ciclo de vida.

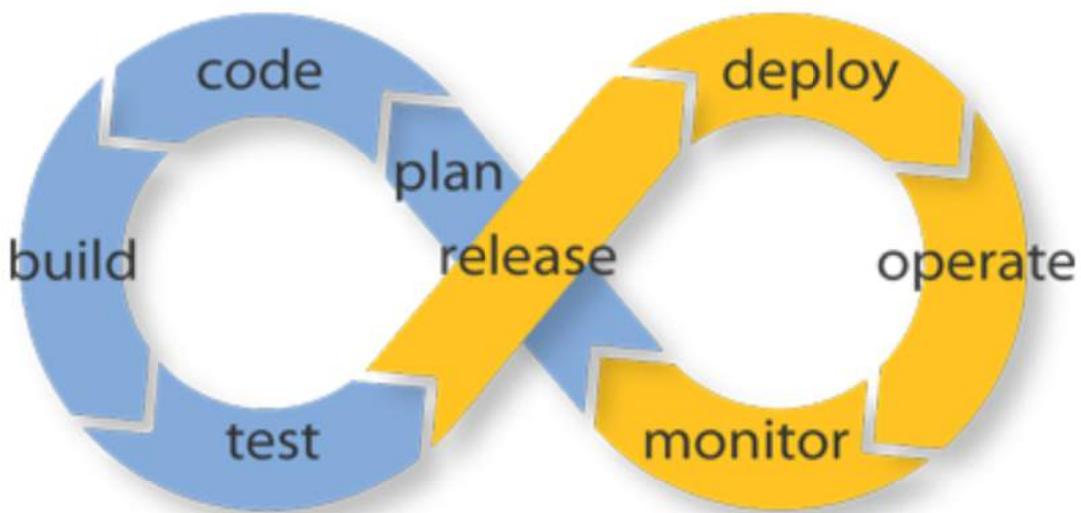
## O sucesso para DevOps

Cultura livre de culpa, que compartilha histórias e desenvolve a empatia, permitindo que pessoas e equipes coloquem em prática suas habilidades de maneira eficaz e duradoura.

Fornecer aplicativos e serviços para o negócio de forma Just-in-Time (JiT).

Assegurar a continuidade dos serviços de TI, por meio de uma abordagem das necessidades do negócio com base nos riscos.

Gerenciar todo o ciclo de vida de aplicativos e serviços, incluindo condições para o final do ciclo de vida.



## Pipeline CI/CD

Para que tenhamos os benefícios DevOps explocarados na sessão “DevOps e seus Benefícios” devemos para tanto implementar um pipeline automatizado, que permita a equipe de desenvolvimento e operação reduzir o **lead-time** (diferença de tempo entre iniciar um processo e ter o produto final; em desenvolvimento definido como a diferença de tempo entre a identificação de uma **user-story** e a utilização da mesma em produção).

Esse pipeline pode ser quebrado em três etapas: **Continuous Integration**; **Continuous Delivery**; **Continuous Deployment**.

# 4LINUX / Pipeline CI/CD

## Integração Contínua (CI)

- ✓ A CI tem por objetivo identificar e corrigir erros mais rapidamente, melhorar a qualidade do software e reduzir o tempo necessário para validar e lançar novas atualizações de software.
- ✓ O código é testado e preparado diretamente (automação) antes de ser liberado para a produção.

## Entrega Contínua (CD)

- ✓ Conjunto de práticas cujo objetivo é garantir que alterações ou novas versões de software, sejam colocadas no ambiente de produção a qualquer momento.

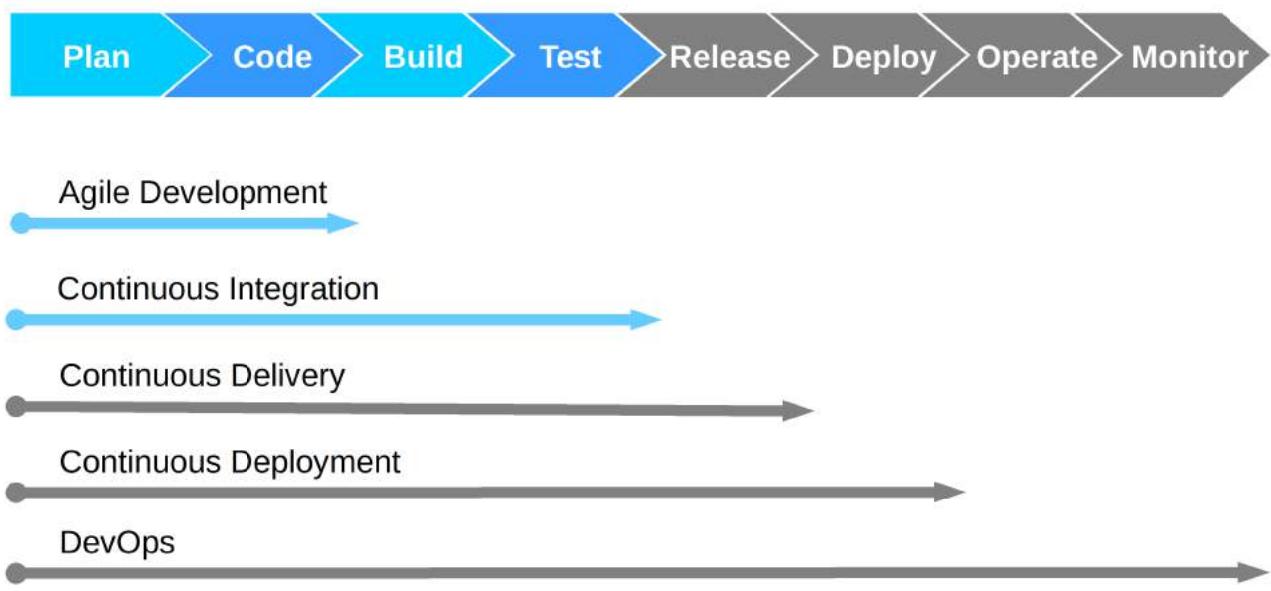
## Implantação Contínua (CD)

- ✓ É uma espécie de extensão da integração contínua, com o objetivo de minimizar o *lead time* em produção realizando deploys automáticos no processo.

**Integração Contínua (CI):** Realizando a utilização de metodologias Agile para realizar implementações e alterações contínuas no código. Nesta etapa utilizando elementos como **User-Stories** e **Testes**, de forma automatizada, preparando uma nova versão do software para realizar a Entrega ou Implementação Contínua.

Após o CI, o passo de Deploy em produção pode ser controlado por duas etapas: Entrega ou Implementação Contínua. A diferença está na automação do deploy para o ambiente de produção. A entrega contínua aguarda uma confirmação “manual” para que a nova versão entre em produção, diferente da implementação contínua que realizará o deploy automaticamente.

Uma ressalva importante a se fazer sobre a limitação da automação é que a mesma não é restringida por elementos técnicos, mas sim por decisões de negócio, como por exemplo uma auditoria externa do seu software.



Durante a fase de desenvolvimento a equipe fará uso de metodologias de desenvolvimento Agile como XP e Scrum. Nessa etapa o grupo de desenvolvimento deve buscar realizar ciclos frequentes de release da aplicação, buscando: Velocidade Estável, Adaptabilidade e Códigos Livres de Bugs. Conceitos como JKK (Ki-Koutei-Kanketsu) e Definição de Pronto auxiliam nesta tarefa.

Assim que um push for realizado para uma solução de versionamento, a fase de integração, com uma solução de CI, deve ser capaz de reconstruir a aplicação e executar todos os testes automaticamente.

## Testes devem ser adaptados na transição para DevOps

- ✓ **Testes automatizados:** como unitários, de componente, sistema, implementação e aceitação.
- ✓ **Testes manuais:** como de apresentação do produto, usabilidade e exploratório.
- ✓ Testes funcionais e não funcionais.
- ✓ Testes são contínuos desde o início do projeto.

## Testes

Constituem atividades multifuncionais que envolvem toda a equipe, devem ser feitos continuamente desde o início do projeto.

Construir qualidade significa criar testes automatizados em vários níveis (unidade, componente e aceitação), executá-los como parte da pipeline de implementação, que será acionada toda vez que uma alteração ocorrer no aplicativo, nas configurações ou no ambiente e, na pilha de software onde o teste funciona.

Os testes manuais também são parte essencial da qualidade da construção em showcases: testes de usabilidade e testes exploratórios que precisam ser feitos continuamente ao longo do projeto.

Construir qualidade, também significa trabalhar constantemente para melhorar a estratégia automatizada de testes.

## Testes devem ser adaptados na transição para DevOps

- ✓ Testes começam a ser definidos antes mesmo dos desenvolvedores iniciarem seu trabalho.
- ✓ Testes também são acionados com mudanças no aplicativo, no ambiente ou na configuração. São regressivos.
- ✓ Testes de aceitação são críticos.

Em um projeto ideal, testadores colaboram com desenvolvedores e usuários para escrever testes automatizados desde o início do projeto.

Esses testes são escritos antes mesmo que os desenvolvedores comecem a trabalhar nos recursos que irão testar. Juntos, os testes formam uma especificação executável do comportamento do sistema. Quando o sistema testado passa nos testes, fica demonstrado que o recurso requerido pelo cliente foi implementado completa e corretamente.

A suíte de testes automatizados, é executada pelo sistema CI sempre que é feita uma alteração no aplicativo, significa que a suíte também serve como um conjunto de testes regressivos, para garantir que tudo que já foi implementado continua funcional.

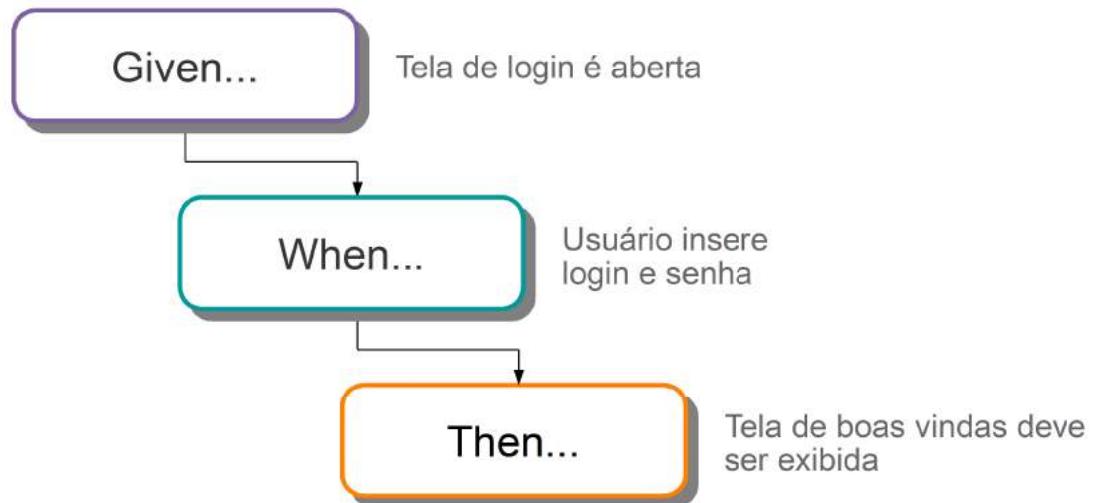
Os testes de aceitação são críticos em um ambiente ágil porque respondem às perguntas:  
para desenvolvedores: "como posso saber quando finalizo?" Para os usuários: "consegui o que desejava?"

### Testes devem ser adaptados na transição para DevOps

- ✓ Testes mostram quando a história atingiu o estágio: "done" (concluída) para os desenvolvedores.
- ✓ Já para os usuários demonstram: "Consegui o que desejava?"
- ✓ Histórias de usuários: podem ser testadas usando o happy path – 'Given-When-Then'

Testes de aceitação devem ser escritos e, idealmente automatizados, antes mesmo da equipe de desenvolvimento iniciar uma história. Os testes de aceitação, como os critérios de aceitação, podem testar todos os tipos de atributos do sistema que está em construção, incluindo: recursos, capacidade, usabilidade, segurança, capacidade de modificação, disponibilidade e assim por diante.

## Teste GWT (Given-When-Then) (Dado-Quando-Então)



Em geral, para cada história ou requisito, existe um único caminho canônico para o aplicativo em termos das ações que o usuário irá realizar. Isso é conhecido como “caminho feliz”, muitas vezes expresso usando a forma “Given” (Dado), algumas características importantes sobre o estado do sistema quando o teste iniciar, “When” (Quando), o usuário executa um conjunto de ações, “Then” (Então), algumas características importantes sobre o novo estado do sistema. Isso às vezes é chamado de modelo para testes: “given-when-then”, “dado-quando-então”, na tradução literal.

No entanto, qualquer caso de uso em todos os sistemas, exceto o mais simples, permitirá variações no status inicial, nas ações a serem realizadas e no status final do aplicativo. Às vezes, essas variações constituem casos de uso distintos, que são conhecidos como caminhos alternativos. Em outros casos, causam condições de erro, resultando no que é chamado de “sad paths”, “caminhos tristes”, na tradução literal.

## Princípio Invest e testes operacionais

Princípios INVEST.

- ✓ Independente, negociável.
- ✓ Valioso, estimativo.
- ✓ Pequeno e testável, com critérios de aceitação.
- ✓ Histórias operacionais: checar se a implementação funcionou.

Histórias bem escritas, obedecem as características do acrônimo INVEST – Independente, negociável, valiosa para o cliente ou usuário, estimável, pequena, (Small em inglês) e testável.

**Independente:** procurar criar histórias relacionadas em uma narrativa única. As relações, com histórias, assim combinadas, tornam-se mais fáceis de gerenciar.

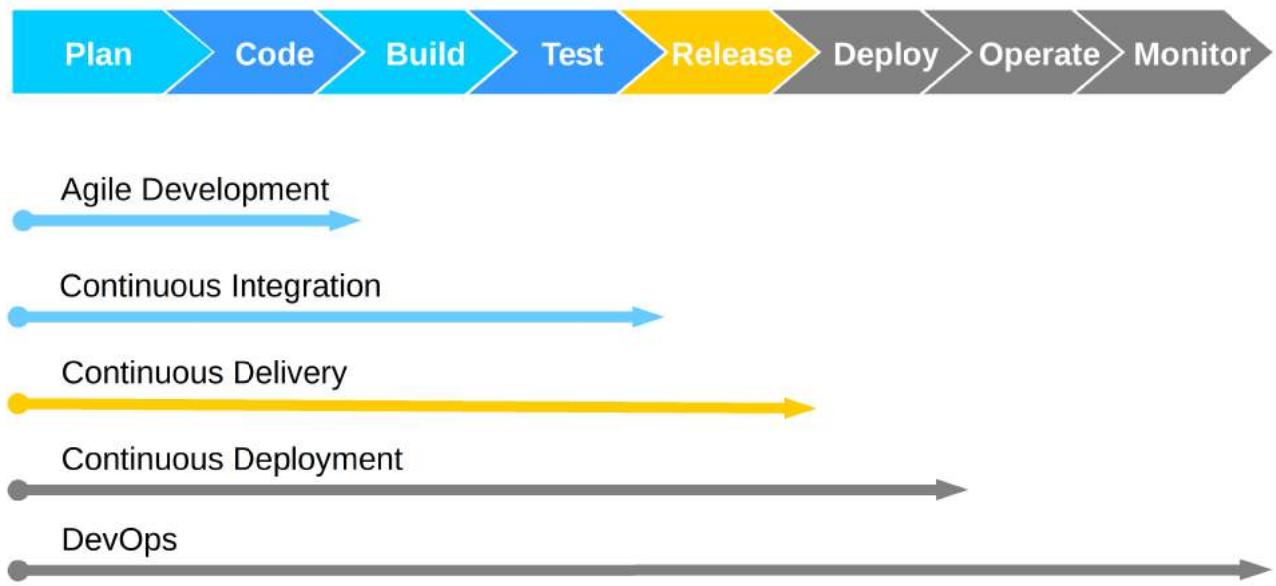
**Negociável:** evitar criar histórias com muitos detalhes, de maneira que os requisitos de entrega, já sejam definidos. As histórias precisam ser genéricas, permitindo mudanças no escopo a cada sprint, devido aos feedbacks recebidos do cliente, possibilitando ajustar continuamente as entregas ao valor do negócio.

**Valiosa para o cliente ou usuário:** escrever histórias de modo que usuários e clientes entendam seu valor e assim possam priorizá-las de acordo. Abstrair termos técnicos.

**Estimável:** deve ser possível estimar o tempo e complexidade de uma história, para que ela possa entrar em um sprint.

**Pequenas:** histórias grandes, chamadas de épicas, devem ser evitadas, porque são difíceis de estimar e executar as tarefas envolvidas. Quebre as em histórias em narrativas menores.

**Testável:** a história deve descrever claramente a entrega desejável, o que permite testá-la por processos automatizados.



### Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Entrega contínua é essencial para DevOps efetivo

- ✓ Continuous Delivery (CD): é a implementação automatizada dos processos de *build*, *integrate*, *tests* e *release* do aplicativo.
- ✓ Garante desempenho, entrega rápida, e conformidade de acordo com os requisitos.
- ✓ Dependendo de uma liberação para o deploy em produção

Usando essas práticas, mesmo grandes organizações com aplicativos complexos podem fornecer novas versões de seu software de forma rápida e confiável.

Isso significa, não só, que as empresas podem obter um retorno mais rápido do investimento, como podem fazê-lo com riscos reduzidos e sem incorrer no custo de ciclos de desenvolvimento longos, ou pior, fornecendo software que não é adequado para o devido propósito.

Para usar uma analogia com o modelo *lean manufacturing*, o software não entregue com frequência, é como o estoque armazenado em um armazém.

Custou dinheiro para fabricá-lo, mas não produz qualquer retorno financeiro, na verdade, também custa dinheiro para armazená-lo.

### Entrega contínua é essencial para DevOps efetivo

- ✓ A entrega rápida permite aos usuários começar o uso do software rapidamente, percebendo seu valor e fornecendo feedback.
- ✓ A manutenção dos releases permite que a entrega de serviços seja tratada da mesma forma.
- ✓ Essencial para permitir a integração contínua.

### Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

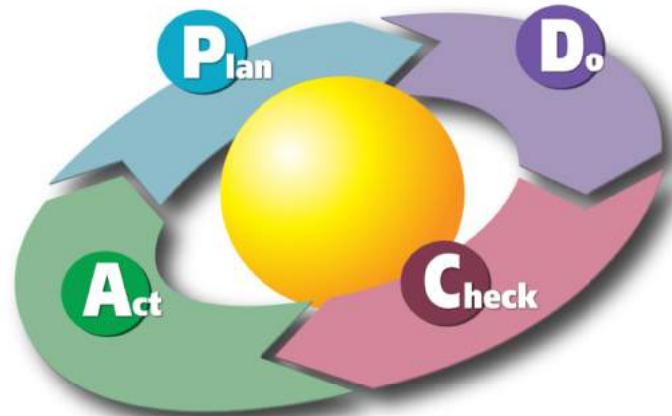
---

---

---

## Como incorporar entrega contínua em um cenário

- ✓ Revisar a situação atual.
- ✓ Focar em um área que é imatura com base nos benefícios.
- ✓ Acordar critérios de aceitação.
- ✓ Planejar e Implementar
- ✓ Prova de conceito em uma pequena área.
- ✓ Revisão de Critérios
- ✓ Repita em outras áreas imaturas



Revisar a situação atual: use o modelo de maturidade para identificar quais áreas são mais imaturas em relação a DevOps. Podem haver diferentes níveis de maturidade.

Focar em uma área mais imatura e mais propícia aos benefícios a serem alcançados com a Entrega Contínua. Decida quais melhorias fazem sentido para sua organização, estime os custos, benefícios e priorizações.

Definir critérios de aceitação, para especificar os resultados esperados e como serão medidos, para que você possa decidir se a mudança foi bem-sucedida.

Em seguida, crie um plano de implementação, **Plan**. Provavelmente é melhor começar com uma prova de conceito. Se esse for o caso, escolha uma parte da organização que está realmente com dificuldades – essas pessoas terão a melhor motivação para implementar mudanças, justamente lá você verá as mudanças mais significativas.

Uma vez que as mudanças tenham sido feitas, **Do**, use os critérios de aceitação que criou para medir se alcançaram os efeitos desejados, **Check**. Faça uma retrospectiva com todos os envolvidos, para definir quão bem foram executadas e quais são as potenciais áreas de melhoria, **Act**.

Repita esses passos e construa seu conhecimento. Execute melhorias de maneira incremental e ao longo de toda a organização.

# Modelo de Maturidade DevOps

Prática	Gestão de compilação e integração contínua	Ambientes e implantação	Gestão de entrega de versão e observância	Testes	Gestão de dados	Gerência de configuração
<b>Nível 3 – Otimização</b> Foco em melhoria de processo	Equipes se reúnem regularmente para discutir problemas de integração e como resolvê-los com automação, feedback mais rápido e melhor visibilidade.	Todos os ambientes são geridos efetivamente. O provisionamento é completamente automatizado. Virtualização é usada se aplicável.	As equipes de operações e de desenvolvimento colaboram regularmente para gerir riscos e reduzir tempo de ciclo.	Rollbacks são raros. Defeitos são encontrados e resolvidos imediatamente.	Existe um ciclo de feedback de entrega e entrega quanto ao desempenho do banco de dados e processos de implantação.	Há validação regular de que a política de gerência de configuração apoia colaboração eficaz, uma processo rápido de desenvolvimento, um processo de gestão de mudança auditável.
<b>Nível 2 – Gerido quantitativamente</b> Processos mediados e controlados	Métricas de compilação são coletadas, disponibilizadas de forma visível e mudanças são feitas com base nelas. Compilações com erros são corrigidas imediatamente.	Implantações gerenciadas e orquestradas. Processos de entrega e de reversão de problemas são testados.	A saúde de ambientes e aplicações é monitorada e gerida proativamente. O tempo de ciclo é monitorado.	Métricas e tendências de qualidade são rastreadas. Requisitos não funcionais são definidos e medidos.	Atualizações de bancos de dados e rollbacks são testadas a cada implantação. O desempenho do banco de dados é monitorado e otimizado.	Os desenvolvedores fazem check-in no trunk pelo menos uma vez por dia. Branches são usados somente para entregas.
<b>Nível 1 – Consistente</b> Processos automatizados aplicados ao longo de todo o ciclo de vida da aplicação	Compilação e testes automáticos são executados para cada check-in. Dependências são gerenciadas. Há reuso de scripts e ferramentas.	Processos completamente automatizados para implantação apertando-se um botão. O mesmo processo é usado para implantar em todos os ambientes.	Gestão de mudança e processos de aprovação são definidos e seguidos. Condições regulatórias e de observância são obtidas.	Existem testes unitários e de aceitação; estes últimos são escritos por testadores. Teste é parte do processo de desenvolvimento.	Mudanças no banco de dados são executadas automaticamente como parte do processo de implantação.	Bibliotecas e dependências são gerenciadas. Políticas de controle de versão são determinadas pelo processo de gestão de mudança.
<b>Nível 0 – Repetível</b> Processo documentado e parcialmente automatizado	Compilação e testes automatizados são regulares. Qualquer binário pode ser recriado do código-fonte usando um processo automatizado.	Implantação automatizada para alguns ambientes. A criação de novos ambientes é barata. Toda a configuração foi externalizada e versionada.	Entregas infreqüentes e árduas, mas confáveis. Rastreabilidade limitada de requerimentos por entregas.	Testes automatizados escritos como parte do desenvolvimento de uma história.	Mudanças para o banco de dados são feitas com scripts automatizados versionados por aplicação.	Controle de versão é usado para tudo que é exigido para recriar o software: código-fonte, configuração, scripts de compilação e implantação, migrações de dados.
<b>Nível -1 – Recessivo</b> Processos que não podem ser repetidos, pouco controlados e reativos.	Processos manuais para compilação de software. Nenhum gerenciamento de artefatos e relatórios.	Processos manuais de implantação binários específicos por ambiente. Ambientes provisionados manualmente.	Testes infreqüentes e sem confiabilidade.	Testes manuais após a implantação.	Migrações de dados sem versionamento e feitas manualmente.	Controle de versão pouco ou não usado.

## Problemas comuns

- ✓ Implementações infrequentes ou “bugadas”.
- ✓ Má qualidade do aplicativo.
- ✓ Processo de integração contínua mal gerenciado.
- ✓ Gerenciamento de configuração deficiente.

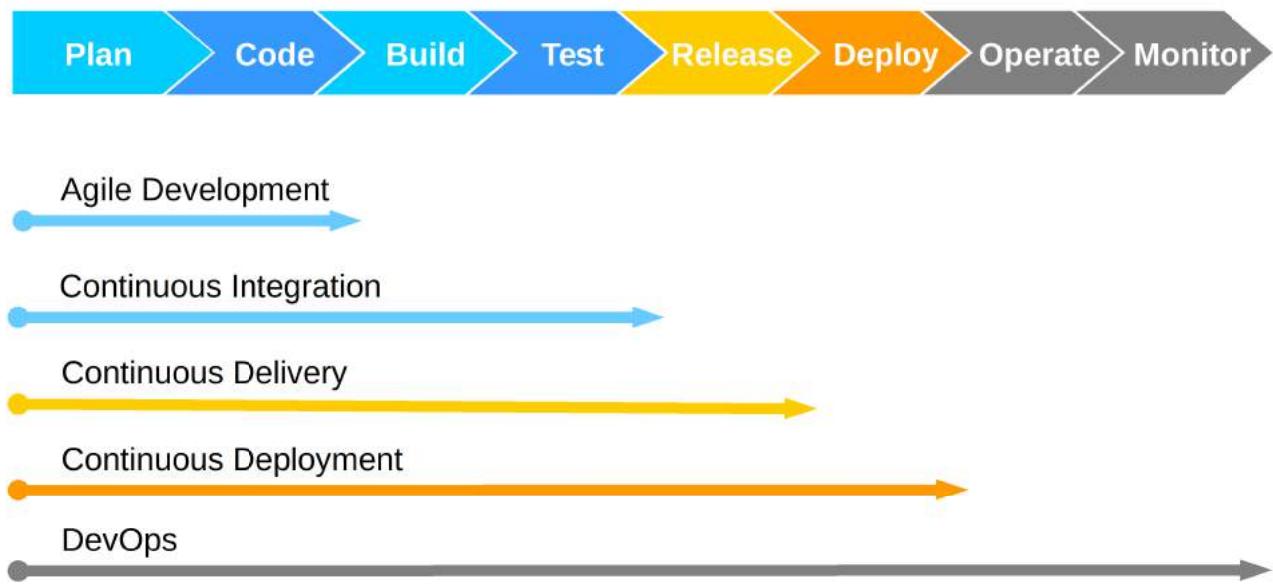
## Problemas comuns

Implementações infrequentes ou “bugadas”: A compilação demora muito e o processo de implantação é frágil.

Má qualidade do aplicativo: as entregas não apresentam a qualidade esperada, erros são recorrentes e tomam muito tempo da equipe para consertá-los.

Processo de Integração Contínua mal gerenciado: chekins infrequentes, commit lento mais de 10 minutos é inaceitável, falta confiança em reverter alterações e falta domínio no processo CI.

Gerenciamento de configuração deficiente: ambientes diferentes, problemas misteriosos em produção, recorrentes rollbacks e patches, colaboração ops e dev ineficiente, falta de monitoramento pró-ativo.



Diferente da Entrega Contínua (*Continuous Delivery*) a Implementação Contínua não é dependente de uma passo manual para colocar a nova versão de um software em produção. Após todas as etapas de Integração Contínua serem satisfeita, o *Deploy* é automaticamente providenciado dentro da infraestrutura, reduzindo o **Lead Time**.

**Implementação Contínua é essencial para DevOps efetivo, e completamente automatizado.**

- ✓ Continuous Deployment (CD): é a implementação automatizada dos processos de *build*, *integrate*, *tests* e *release* e **DEPLOY** do aplicativo.
- ✓ **NÃO** Dependendo de uma liberação para o deploy em produção

### Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

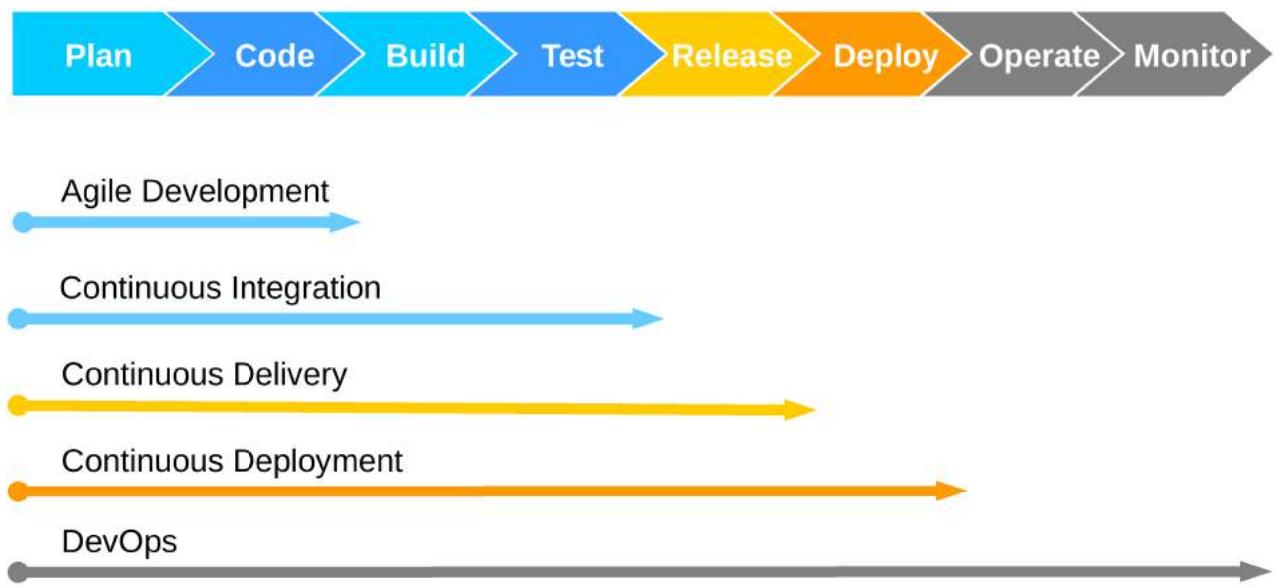
---

---

---

---

---



### Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---



## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Objetivos da Aula

- ✓ História.
- ✓ Vantagens.
- ✓ Servidores Git.

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Git x Linux



Como muitas coisas na vida, o Git nasceu de uma criatividade destrutiva e muita controvérsia.

O kernel Linux é um projeto open source de escopo gigantesco. Durante os anos de 1991 a 2002, as alterações no kernel foram realizadas com a troca de patches de correção e arquivos. A partir de 2002, passou-se a utilizar um DVCS (Distributed Version Control System) chamado BitKeeper.

Em 2005, a parceria com a comunidade de desenvolvedores do Kernel Linux e os desenvolvedores do BitKeeper foi quebrada, levando o status, de livre de pagamento, das ferramentas a ser revogado. Isso levou a comunidade e principalmente o desenvolvedor do Kernel, Linus Torvalds, a produzir um nova ferramenta utilizando toda experiência adquirida com o uso do BitKeeper. As premissas da nova ferramenta seriam:

Velocidade.

Design simplificado.

Forte suporte ao desenvolvimento não linear (fazendo uso de branches).

Totalmente distribuído.

Capaz de suportar grandes projetos (kernel) mantendo a simplicidade e velocidade.

Desde seu nascimento em 2005 o Git vem evoluindo, para se tornar um sistema simples de uso, sem perder as qualidades acima mencionadas.

## História e mercado

**1991/2002**

O kernel do linux não usava nenhum Sistema de Controle de Versão.

**2002/2005**

Início do uso do BitKeeper.

**2005**

Criação do Git.

A história começa em 2002, quando o kernel Linux começou a utilizar um sistema chamado BitKeeper para o controle de versão. Até então eles versionavam o código usando patches de arquivos compactados.

Em 2005 a empresa responsável pelo desenvolvimento comercial do BitKeeper começou a cobrar pelo uso. A comunidade então resolveu desenvolver o próprio versionador baseado nas lições aprendidas no BitKeeper.

## O que o Linux tem haver com isso?

GIT = Gíria britânica para pessoas teimosas, que sempre acham que têm razão.

“I'm an egotistical bastard, so I name all my projects after myself. First Linux, now git”.

Linus Torvalds



## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## O que o Linux tem haver com isso?

As premissas do projeto.

- ✓ Desempenho.
- ✓ Totalmente distribuído.
- ✓ Design simples.
- ✓ Desenvolvimento não linear, **branches em paralelo**.
- ✓ Ser eficiente com grandes projetos. **Kernel do Linux**.

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Vantagens por ser distribuído.

- ✓ Trabalhar offline.
- ✓ Commit constante, sem medo.
- ✓ Velocidade.
- ✓ Branchs locais.
- ✓ Desenvolvedor decide quando ou o que deve subir.
- ✓ Vários backups.

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Seguro

- ✓ Armazena hash dos arquivos.
- ✓ Garante integridade dos arquivos e das versões.
- ✓ GIT garantiu integridade dos repositórios em 2011.

Kernel.org invadido.

- ✓ Branchs locais.
- ✓ Desenvolvedor decide quando ou o que deve subir.
- ✓ Vários backups.

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## História e mercado

facebook.

Google

NETFLIX



Microsoft

twitter

LinkedIn

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Gerenciadores de projetos

- ✓ **Github (cloud)**: lançado em 2008, desde então desenvolvedores usam para hospedar seus projetos.
- ✓ **Gitlab (cloud e local)**: é um gerenciador de repositório Git para Web, também dispõe Wiki e rastreamento de Issues.
- ✓ **Bitbucket (cloud)**: consiste em um serviço de hospedagem de projetos, controlado através do Mercurial, um sistema de Controle de versões distribuído, porém, também suporta git.



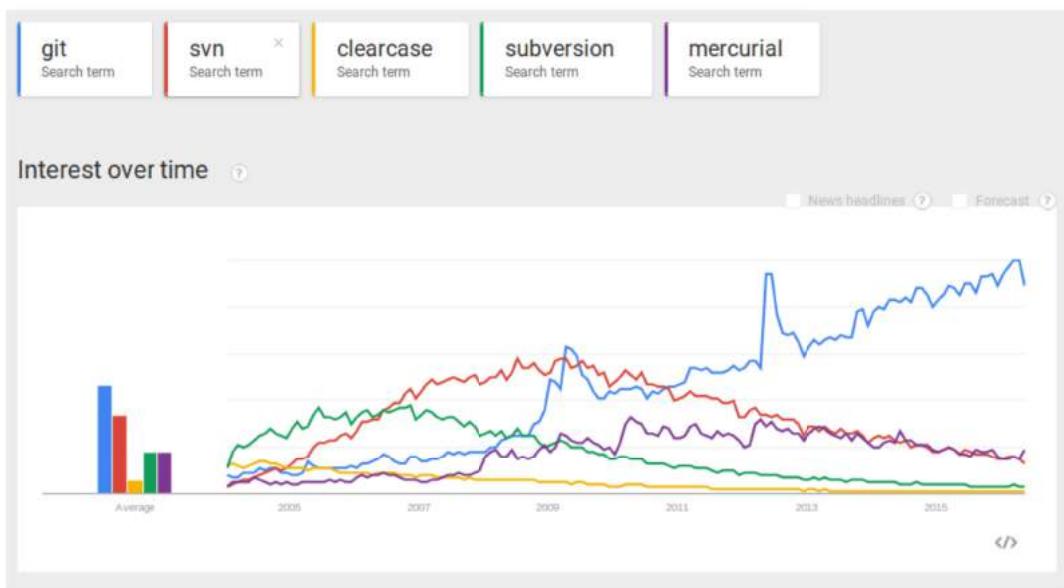
### Introdução

Github: free para projetos públicos e open source. Para projetos privados é necessário ter uma conta paga.

Gitlab: pode ser instalado de forma local ou utilizando a sua Cloud própria de forma totalmente free, 100% atualizada. Versão comunidade e versão enterprise. Versão enterprise possui algumas facilidades de integração com AD/LDAP, exportar CSV das Issues, Code Quality, entre outras vantagens.

Bitbucket: free até 5 usuários

## Tendência



## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Objetivos da Aula

- ✓ Instalar Git.
- ✓ Criar projeto.
- ✓ Configurar o Git.
- ✓ Primeiro commit.
- ✓ Interagir com o Git.

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Instalando o Git

1 Instalação do Git na distribuição Debian/Ubuntu:

```
# apt-get install git
```

2 Instalação do Git na distribuição CentOS:

```
# yum install git
```



## Instalando o Git

A instalação do Git pode ser realizada simplesmente utilizando o gerenciador de pacotes da distribuição, no caso do Debian deve-se utilizar o APT, para CentOS o YUM.

## Iniciando um repositório

Qualquer diretório pode se tornar um repositório Git, para isso, basta inserir o comando:

1 Crie um novo diretório:  
# mkdir projeto

2 Entre no diretório:  
# cd projeto

3 Inicie o repositório:  
# git init

Ao digitar esse comando, será criado um diretório oculto chamado: **.git**, em qualquer divisão de disco.

## Iniciando repositórios

Este comando irá criar um diretório oculto chamado **.git**. Esse diretório é a base de dados do Git, constitui o repositório em si. Nele serão guardadas as informações de **commits**, **tags**, **branches**, ou seja, todas as informações referentes ao repositório.

Os comandos que aprenderemos do git, manipularão o conteúdo desta pasta, por isso não precisaremos mexer diretamente nela.

## Configurando o Git

Salvando um novo estado no Git, algumas informações acompanharão esse processo, tais como: e-mail, nome, data, hora e mensagem de identificação. Alguns dados o Git gera sozinho, outras informações, devemos informar:

1 Configurando o nome:

```
# git config --global user.name "Gustavo"
```

2 Configurando o e-mail:

```
# git config --global user.email "gustavo@4linux.com.br"
```

Com o comando **git config**, salvamos dados nas variáveis de configuração do Git. No nosso caso, informações de nome e e-mail.

O parâmetro **--global** diz ao git que salvaremos esses valores no arquivo **~/.gitconfig** , essas configurações serão utilizadas para todos os outros projetos daquele usuário. Existe mais outro parâmetro, o **--system**, que salva as informações em **/etc/gitconfig** , sendo usado para todos os projetos de todos os usuários.

A ausência desses parâmetros irá gravará as informações no **.git/config**, sendo assim, válidas apenas para aquele projeto. Podemos forçar a utilização do **.git/config** passando o parametro **--local**.

## Configurando o Git

Outras configurações podem ser realizadas. Vamos configurar o editor padrão do Git, que aparecerá sempre que salvarmos uma nova versão do projeto:

1 Configurando editor padrão:

```
# git config --global core.editor vim
```

Podemos ver uma lista com as configurações já setadas:

2 Listando todas as configurações:

```
# git config --list
```

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Primeiros passos

Agora que temos nossas informações já configuradas, vamos iniciar o versionamento do conteúdo deste diretório:

- 1 Vamos criar um arquivo qualquer:  
`# touch arq01`

- 2 Adicionar o arquivo em stage:  
`# git add arq01` ou `# git add *`

- 3 Para visualizar o estado atual do repositório:  
`# git status`

- 4 Para gravar a nova versão do repositório:  
`# git commit -m "Meu primeiro commit"`

O comando **git add** adiciona o arquivo arq01 em uma área chamada **stage**.

Depois que temos o arquivo em stage, podemos gravar o estado dele com o comando **git commit**. Este comando salva o estado dos arquivos que estão na área de stage.

O comportamento é semelhante a bater uma fotografia do repositório naquele exato momento. O parâmetro **-m**, atribui uma mensagem ao commit, servindo para identificar mais facilmente o que foi realizado naquela alteração.

Já o comando **git status**, mostra o estado do repositório naquele momento. Trata-se de um comando bem importante, que usaremos muito em nosso curso.

## Modificar Último Commit

1 Crie dois arquivos para o teste e adicione para commit:

```
# touch arq3 arq4 arq5  
# git add arq3 arq4
```

2 Adicione um comentário no commit:

```
# git commit -m "Commit incompleto"
```

3 Adicione os arquivos para envio ao repositório:

```
# git add arq5
```

4 Envie os arquivos faltantes:

```
# git commit --amend
```

## Modificar último Commit

Ao executar o passo 2, o git retornará uma mensagem informando que os arquivos arq3 e arq4, passaram por commit indo para o repositório local.

Para modificar o commit deve-se adicionar o arquivo arq5 no passo 3, utilizando a opção --amend para modificar o último commit, passo 4.

## Clonando um repositório

Podemos também clonar um repositório que já existe na web:

1

Clonando um repositório:

```
# git clone https://github.com/Gamboua/silex-sample.git
```

2

Para clonar e dar um nome específico ao novo diretório:

```
# git clone https://github.com/git/git.git nomedodiretorio
```

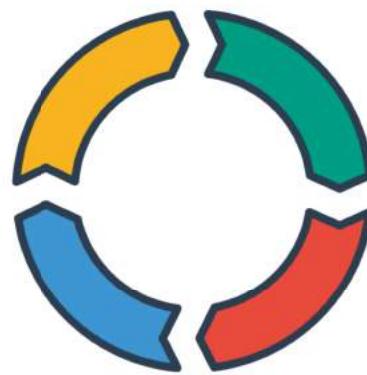
O comando git clone, copiará todos os arquivos do repositório remoto para a sua máquina. Mais a frente, assimilaremos melhor os repositórios remotos.

## Clonando um repositório

Com a opção clone, realizaremos uma cópia do repositório remoto para o diretório local, podendo ser com o mesmo nome ou, um nome diferente caso especificado. Um detalhe importante, a clonagem trará a configuração de repositórios remotos (git remote), que será vista mais adiante.

## Ciclo de vida dos arquivos

Para entender melhor como o Git funciona, vamos verificar os estados dos arquivos:



- ✓ untracked
- ✓ unmodified
- ✓ modified
- ✓ staged

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Untracked files

Arquivos marcados como **untracked**, não estão monitorados pelo Git. São considerados novos no repositório:

1 Vamos criar um arquivo qualquer:  
# touch arq05

2 Para checar o status do arquivo:  
# git status

Arquivos não monitorados:

(utilize "git add <arquivo>..." para incluir o que será submetido)

arq05

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Staged files

Para que o Git inicie o monitoramento do arquivo, vamos usar o comando **git add**. Este comando altera o arquivo para **staged**:

- 1 Para adicionar o arquivo em stage:  
# git add arq05

Mudanças a serem submetidas:  
(utilize "git rm --cached <arquivo>..." para não apresentar)

new file: arq05

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Staged files

Os arquivos marcados como **staged** são novos ou modificados. Serão salvos no commit, ou seja, as mudanças que consistirão a próxima versão. Posteriormente os arquivos vão para o estado **unmodified**.

- 1 Para adicionar o arquivo em stage:

```
# git commit -m "Mensagem do commit"
```

```
[master (root-commit) 7d9b9fb] Mensagem do commit
 1 file changed, 0 insertions(+), 0 deletions(-)
  create mode 100644 arq05
```

O comando **git commit**, grava as alterações criando uma nova versão. É como se tirássemos uma fotografia do projeto naquele exato momento.

Sempre precisamos adicionar uma mensagem ao commit, servirá como identificação para saber o que foi feito naquela versão. Criar mensagens com boas descrições do que foi realizado, facilitará um pouco sua vida. É possível inserir a mensagem com o parâmetro **-m**, mas se o parâmetro não for passado, o git abrirá o editor padrão (`git config --global core.editor vim`) para que uma mensagem seja inserida.

## Modified files

Arquivos **modified** já estavam monitorados pelo Git, tiveram alguma alteração desde sua última versão:

1 Para adicionar o arquivo em stage:  
# echo "First Line" >> arq05

2 Vamos checar o estado do repositório:  
# git status

```
No ramo master
Changes not staged for
commit:

modified:   arq05
```

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Modified files

Vamos reaplicar as ações realizadas: passar o arquivo para **staged** com o comando **git add**, em seguida, salvar com o comando **git commit**:

- 1 Adicionando em stage  
# git add arq05

- 2 Vamos checar o estado do repositório:  
# git commit -m "Mensagem do commit"

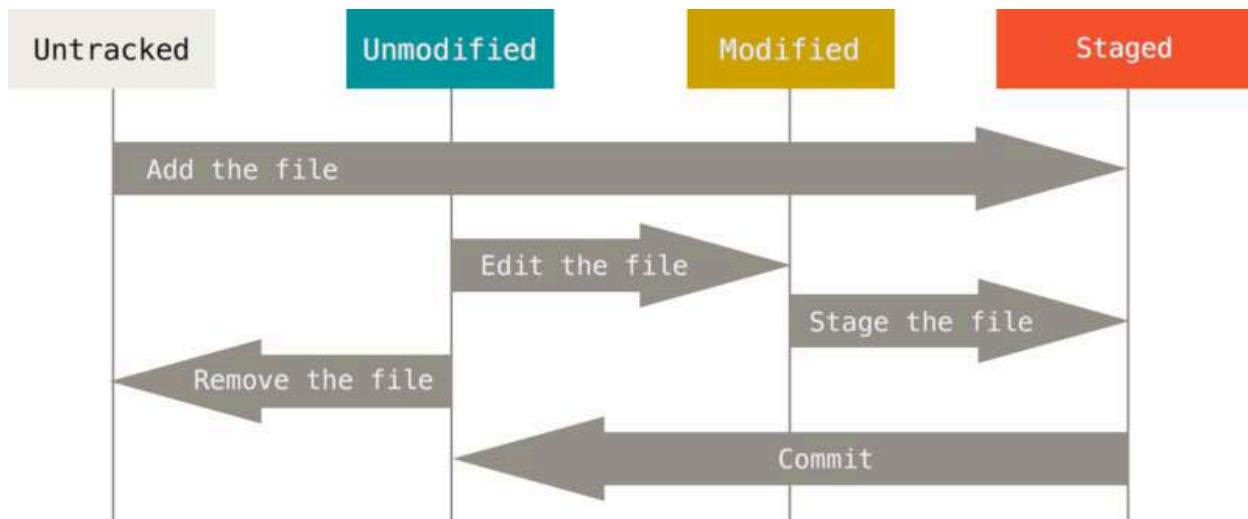
```
[master d095be6] Mensagem do commit
 1 file changed, 1 insertion(+)
```

Essa sequência de comandos (**git add** e **git commit**), é muito comum no uso diário do Git. Existe uma forma simplificada, **git commit -a -m "Mensagem do commit"**.

Com o parâmetro **-a**, o Git adiciona primeiro os arquivos em modified em seguida salva a versão. Entretanto, esse parâmetro não adiciona como **staged**, arquivos que estavam como **untracked**, apenas arquivos como **modified**.

## Relationship between File States

Com o gráfico abaixo podemos melhor entender o estados de cada arquivo no fluxo de operação do git.



<https://git-scm.com/book/en/v2/Git-Basics-Recording-Changes-to-the-Repository>

## Desfazendo as coisas

Nem sempre tudo acontece como gostaríamos. Vamos examinar os comandos para desfazer as ações que realizamos:



- ✓ **reset**
- ✓ **revert**
- ✓ **checkout**

### Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## git checkout

Uma das principais funções do **git checkout** é desfazer as mudanças nos arquivos.

- 1 Vamos alterar o arquivo:  
# echo "New Line" >> arq05
- 2 Para checar o status do arquivo:  
# git status
- 3 Vamos remover as mudanças no arquivo:  
# git checkout arq05
- 4 Vamos ver como ficou o repositório:  
# git status

Arquivos não monitorados:  
(utilize "git add <arquivo> ..." para incluir o que será submetido)

Arq05

No ramo dev  
Your branch is up-to-date  
with 'master'.  
nada a submeter, diretório de trabalho vazio

O **git checkout** possui três funções distintas: desfaz mudança em arquivos, retorna no tempo em um commit específico, muda a branch. O primeiro caso, corresponde ao apresentado no slide, retornamos um arquivo para o estado de modified para unmodified.

O segundo caso, pode ser feito para voltarmos no tempo a um commit específico, por exemplo: caso queira saber como estavam os arquivos a X commits atrás.

O terceiro e último caso, veremos melhor no próximo capítulo onde aprenderemos a trabalhar com branches.

## git reset

Pode ser usado para remover commits, mais comum a utilização para remover arquivos marcados como **stage**.

1 Vamos alterar o arquivo:

```
# echo "New Line" >> arq05
```

2 Para adicionar o arquivo em stage:

```
# git add arq05
```

3 Para ver o estado do arquivo:

```
# git status
```

4 Para remover o arquivo de stage:

```
# git reset arq05
```

Mudanças a serem submetidas:

(use "git reset HEAD  
<file>..." to unstage)

modified: arq05

Unstaged changes after reset:

M arq05

Após executar o passo 4, as alterações do arquivo arq5 estarão prontas para remoção do repositório local, bastando para isso rodar o comando **git checkout arq5**.

## git reset

O reset pode ser usado para desfazer um commit. **Muito cuidado**, pois ele também remove o histórico:

1 Vamos alterar o arquivo:  
# git reset HEAD~3

2 Para checar o status do arquivo:  
# git status

```
Unstaged changes after reset:  
M arq05  
  
No ramo master  
Changes not staged for  
commit:  
modified: arq05
```

Neste caso, voltamos 3 commits na linha do tempo. O histórico desses 3 commits será perdido, mas as modificações permanecerão. Não conseguimos remover um commit específico, tudo que aconteceu depois do commit selecionado será apagado.

Este comando não é recomendado para remover commits que já estão no repositório remoto, apenas commits locais.

## git revert

Desfaz um commit de forma segura. Não o remove do projeto, cria um novo, porém, de uma versão do projeto sem commit removido.

1

Vamos alterar o arquivo:  
# git revert 8c7210d



O **git revert** remove um único commit da linha do tempo, o faz de forma **segura**. Na verdade, não deleta o commit do projeto, **cria uma nova versão** do projeto sem aquele commit.

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

### Objetivos da Aula

- ✓ Branches
- ✓ Tags
- ✓ Merge
- ✓ Conflitos
- ✓ .gitignore
- ✓ Hooks
- ✓ Logs

### Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

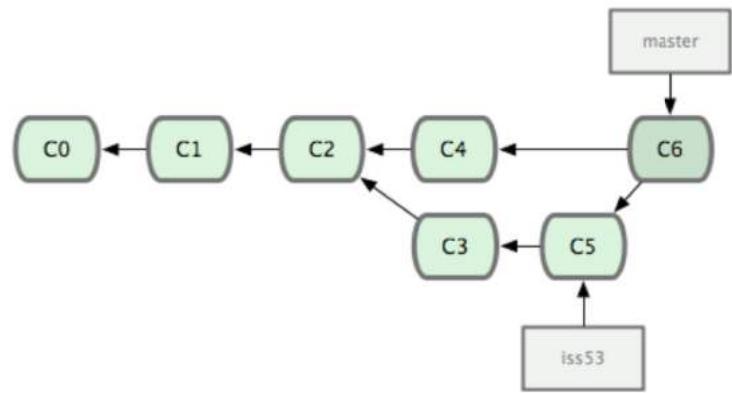
---

---

---

---

---



## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

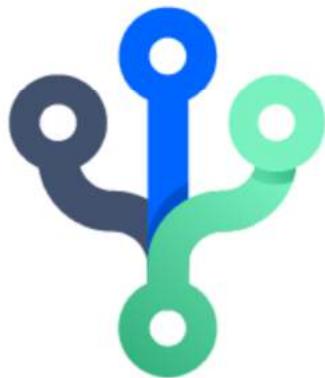
---

---

---

---

---



### **Criar uma branch no Git significa:**

*“...divergir da linha principal de desenvolvimento e continuar a trabalhar sem bagunçar essa linha principal”*

Fonte: <https://git-scm.com/book/pt-br/v1/Ramificação-Branching-no-Git>

## Branches

Realizando a associação com árvores, branches (ramos), são linhas que divergem do tronco principal de desenvolvimento. Em novos ramos, realizamos o desenvolvimento de componentes que serão combinados com o projeto principal.

- ✓ No git, todo o desenvolvimento é efetivado em **branches**.
- ✓ Novas features.
- ✓ Criar fluxos de desenvolvimento, **pipelines**.
- ✓ Softwares que possuem múltiplas versões suportadas.
- ✓ Podem ser locais ou remotas.

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

Tudo no Git é realizado através de branches. A branch principal é a master. Automaticamente criada sempre que um projeto Git é clonado ou iniciado a partir de um diretório existente:

- 1 Listar branches disponíveis:  
`# git branch`

- 2 Para criar uma nova branch:  
`# git branch dev`

- 3 Para mudar de uma branch para outra:  
`# git checkout dev`  
`# git checkout master`

No exemplo do slide, inicialmente estamos na branch. Em seguida, utilizamos o comando **git branch** para criar uma branch chamada develop (dev). Ao executar o comando **git branch** novamente, observamos, que embora o branch develop tenha sido criado, ainda estamos na branch master. A branch develop por enquanto, é uma cópia idêntica da branch master.

Depois, o comando **git checkout** nos trocou para a branch desejada. Agora, ao executar o comando **git branch**, verificamos a mudança para a branch develop. Podemos mudar qualquer arquivo dentro da branch develop, isso não afetará nada na branch master, ela sempre estará intacta.

Não há limites para essas ramificações. Podemos gerar branches a partir de outras e, com isso, gerando versões diferentes de um mesmo projeto. Utilizar branches ajuda muito quando precisamos criar uma nova feature ou, corrigir algo em específico sem modificar a branch principal.

Conforme as branches se acumulam, podemos removê-las após o uso, assim evitamos poluir o seu repositório:

1 Removendo uma branch:

```
# git branch -d dev
```

2 Forçando a remoção de uma branch:

```
# git branch -D dev
```

3 Excluir do Repositório:

```
# git push origin :dev
```

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Exemplo para ambientes de integração contínua

- ✓ Branch Master - Versão corrente, pronto para produção.
- ✓ Branch Stage - Versão de homologação, código testado.
- ✓ Branch Develop - Branch onde são realizados os commits.

### Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

Através das **Tags**, há possibilidade de marcar pontos importantes durante o projeto. Normalmente as pessoas utilizam esta funcionalidade para marcar versões de release (v1.0, v1.1, v2.0).



No decorrer do nosso desenvolvimento, é natural que tenhamos de destacar alguns pontos específicos no código. Normalmente utilizamos as tags para destacar pontos de release.

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

O Git possui duas formas de Tag: *Annotate* (Anotada) e *Light-Weight* (Peso-Leve)

**Anotada:** cria um novo objeto na base do git com informações do Usuário que a criou, como e-mail, nome, e data de criação.

**Peso-Leve:** somente adiciona o tag a um commit já existente.



## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

Por padrão, quando criamos uma tag, é gerada uma referência ao commit atual:

1

Criando uma tag:

```
# git tag -a 1.0 -m "Versao 1.0"
```

Também podemos criar uma tag para um commit específico:

2

Criando uma tag anotada:

```
# git tag -a 1.1 -m "Mensagem do commit" 8cbg67h
```

O parâmetro **-a** define a tag como anotada e o **-m** a mensagem que acompanhará essa tag.

O parâmetro **-m** serve para dar uma mensagem a Tag. Essas tags com mensagens são automaticamente definidas como tags anotadas, não necessitando o parâmetro **-a**. Podemos ter mais de uma tag apontando para o mesmo commit, mas não podemos ter tags com nomes repetidos. Tags peso-leve não possuem mensagem e não utilizam o parâmetro **-a**.

As tags são muito utilizadas para criar versões do projeto (1.0.0, 1.0.1, etc.). Podemos criar tags a qualquer momento de qualquer commit. Servem também para dar um “apelido” ao commit, tornando assim mais fácil encontrá-lo. É possível utilizar esses apelidos como referência ao commit, em outros comandos, por exemplo: **git revert 1.2**.

Podemos listar as tags que foram criadas no projeto:

1 Listando tags:

```
# git tag
```

Podemos também, excluir tags:

2 Excluindo tag:

```
# git tag -d 1.1
```

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

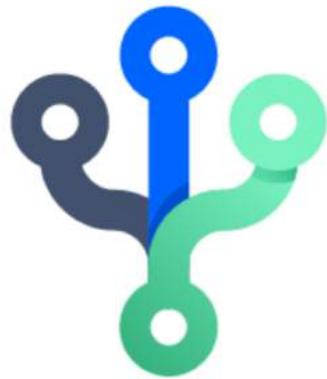
---

---

---

---

---



- ✓ Branches: ramos de desenvolvimento não linear.
- ✓ Tags: fotografias de estados de desenvolvimento, branch imutável.

### Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

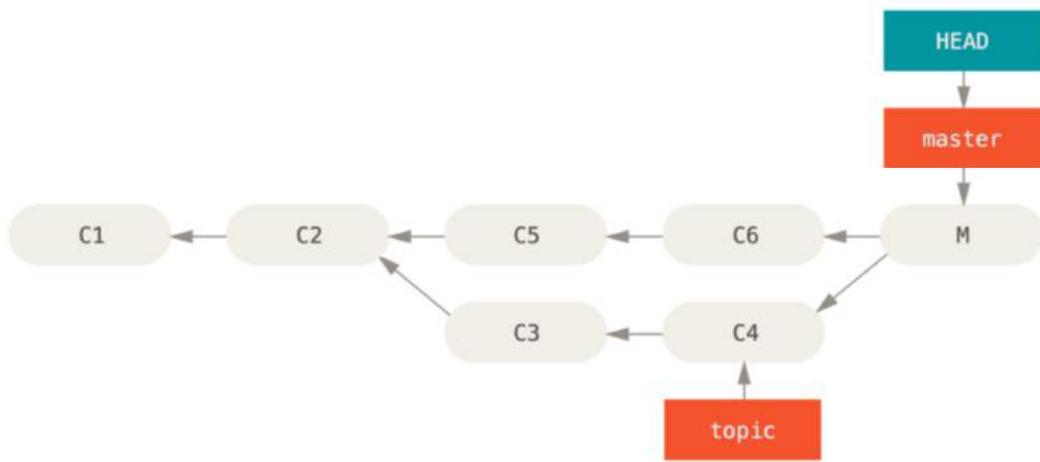
---

---

---

---

---



## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## git merge

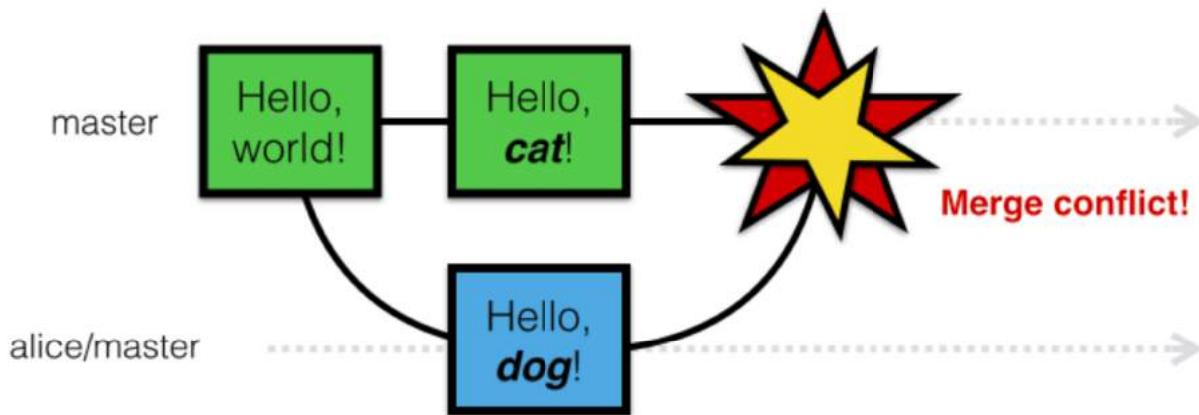
Vamos criar uma nova Branche para simular os Merges

```
1 # git checkout -b develop
2 # touch arq06
3 # git add arq06
4 # git commit -m "Novo Commit"
5 # git checkout master
6 # git merge develop
```

Sempre que criamos uma nova branch, esta, inicia como uma cópia da branch atual. Tudo que for feito nessa nova branch e passar por commit, produzirá uma diferença em relação a branch copiada. O git merge traz esses commits a frente e os aplica na branch atual.

## Conflitos de merge

Podemos juntar branches. O que acontece se juntarmos branches que possuem alterações nos mesmos arquivos?



<https://developer.atlassian.com/blog/2015/01/a-better-pull-request/merge-conflict.png>

## Conflitos de Merge

O conflito de merge ocorre quando dois arquivos de mesmo nome possuem alterações nas mesmas linhas. Essa condição exige alteração do arquivo manualmente, para decidir qual modificação deve entrar passar por commit.

## Conflitos de merge

Caso tenha conflitos de arquivos em Branches na hora do merge, você deverá resolver os conflitos manualmente:

```
Mesclagem automática de arq
CONFLITO (conteúdo): conflito de mesclagem em arq
Automatic merge failed; fix conflicts and then
commit the result.

<<<<< HEAD
Alteração na branch master
=====
Alteração no mesmo arquivo na dev
>>>>> dev
```

### Explicando melhor a saída apresentada no slide

Modificamos o arq06.

Adicionamos e passamos a mudança por commit.

Mudamos para uma nova branch.

Modificamos o mesmo arq06.

Adicionamos e passamos a mudança por commit naquela branch.

Retornamos para a branch máster.

Fizemos o merge e recebemos informação do conflito.

O Git modificou o arquivo, como podemos ver no cat.

A marcação com HEAD diz qual a versão está na branch atual (master).

A marcação debaixo, diz a qual versão está na branch que tentamos trazer.

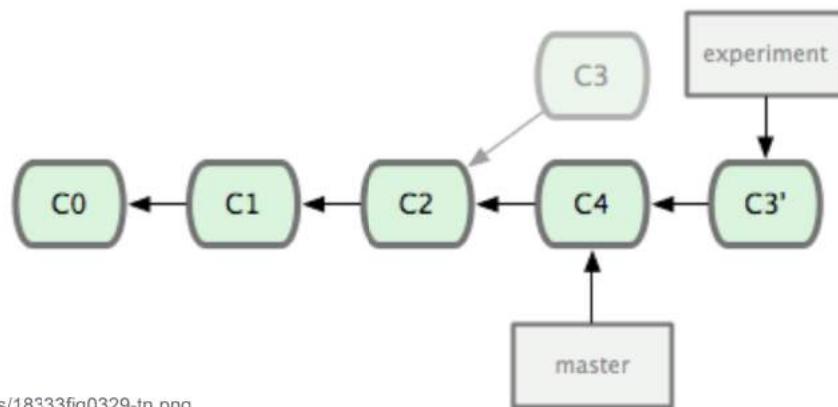
Mexemos no arquivo o deixando do jeito correto.

Adicionamos e passamos a mudança por commit.

Pronto! Conflito solucionado.

## Rebase

O **rebase** é uma forma diferente de fazer merge. A principal diferença é que um histórico, será mantido mais tempo, como se todo o trabalho tivesse sido feito de forma linear.



<https://git-scm.com/figures/18333fig0329-tn.png>

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## git rebase

O comando git rebase jogará os commits diferentes para uma determinada Branch.

1

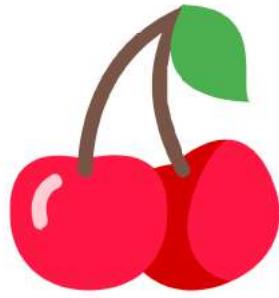
De uma branch diferente da master, enviaremos os commits diferentes:

```
# git rebase master
```

First, rewinding head to replay  
your work on top of it...  
Applying: added staged command

Com o **git merge**, o Git cria um novo commit que aponta para a versão da outra branch com as alterações. No caso do rebase, este não cria um novo commit, mas insere o commit no início da branch.

## Cherry-Pick



O cherry-pic é outra forma de juntar alterações de branches distintas. Ele permite informar um commit específico que queremos inserir em algum branch.

### Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## git cherry-pick

O comando git cherry-pick obterá alterações de um ou mais commits especificados, fará o merge na branch atual.

1

Para aplicar um ou mais commits na branch atual:

```
# git cherry-pick 93c5535
```

```
[master f6885c5] Alteração arq1
Date: Fri Sep 30 16:12:13 2016 -0300
1 file changed, 3 insertions(+)
```

Nesse ponto o **cherry-pick** se parece muito com o rebase, visto anteriormente. Uma diferença entre os dois, o cherry-pick cria um novo commit com a nova alteração, já o rebase, insere a alteração no início da branch.

Funciona como o merge, porém, com o extra de possibilitar escolher exatamente os commits que serão inseridos. Sendo assim, o seu histórico pode ficar menos legível se comparado com o rebase.

Outra coisa interessante sobre o **cherry-pick**, além de aplicar commits em outras branches, também podemos aplicar commits vindo de outros repositórios remotos.

Importante.

- ✓ Cada projeto deve ter seu .gitignore
- ✓ Arquivo oculto na raiz da aplicação chamado gitignore.
- ✓ Especifica intencionalmente arquivos que não serão rastreados pelo Git.
- ✓ Dados sensíveis.



O gitignore especifica arquivo em estado untracked que o Git deve ignorar. Qualquer arquivo que se apresente em estado tracked continuará como tal, precisando ser removido manualmente.

Sintax arquivo gitignore

Qualquer linha vazia ou começando com “#” ou espaço, é desconsiderada pelo Git. No caso de um padrão começando com “#”, o mesmo deve ser escapado com “\”, bem como espaço.

“!” — Deve ser utilizado para negar um padrão, tornando-o valido novamente. Não valendo para diretórios.

“foo/” — Caso o padrão termine em “/” o git irá considerar somente o diretório foo.

“foo” — Nesse caso será tratado como um padrão a ser verificado no level mais alto do diretório onde o .gitignore se encontra.

“\*\*/” — Casa qualquer diretório. Por exemplo: se especificado “\*\*/foo” será válido para “foo/” e “\*\*/foo/bar”, sendo válido para qualquer diretório “bar” antecedido por “foo”.

“/\*\*” — Casa todo o conteúdo dentro do diretório.

“a/\*\*/b” — Casa qualquer conteúdo entre os diretório existentes entre a e b.

1 Criar arquivo gitignore

```
# vi .gitignore  
arq05  
# git add .gitignore
```

2 Remover arquivo localmente e do Cache do Git

```
# rm arq05  
# git rm --cached cache.properties
```

3 Realizar o commit

```
# git commit
```

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Client-Side Hooks

Esses scripts são executados sempre que algum evento acontece no repositório local.



- ✓ **pre-commit**: executa sempre que um snapshot está prestes a ser commitado.
- ✓ **prepare-commit-msg**: atua antes que o editor de mensagem do commit seja executado.
- ✓ **commit-msg**: atua após a mensagem ser inserida no commit.

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

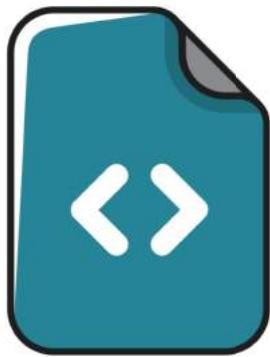
---

---

---

## Client-Side Hooks

Esses scripts são executados sempre que algum evento acontece no repositório local.



- ✓ **post-commit**: útil para notificar outra ferramenta.
- ✓ **post-merge**: prepara ambiente após merge.
- ✓ **pre-push**: efetua validação antes de enviar.

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

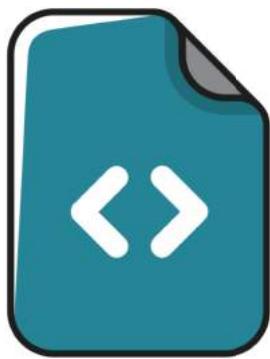
---

---

---

## Server-Side Hooks

Esses scripts são executados sempre que uma mudança chega ao repositório remoto.



- ✓ **pre-commit:** esse script rodará sempre que um push chegar ao repositório.
- ✓ **post-receive:** executado ao final do push.
- ✓ **update:** Parecido com o pre-receive, entretanto é usado para cada uma das branches e tags que estão subindo no push.

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Logs e Histórico

Depois de criar commits ou clonar um repositório, com commits já feitos, nada mais natural que voltar e ver o que foi realizado. Veremos o histórico no Git.



### git log

Comando para mostrar o histórico do repositório. Uma ferramenta poderosa para filtrar e formatar a saída das informações.

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Formatando saída

Formas de mudar a saída do comando padrão `git log`:

- 1 Para resumir o conteúdo mostrado no `git log` padrão:  
`# git log --oneline`
- 2 Para exibir a quais tags e branches pertecem aqueles commits:  
`# git log --decorate`
- 3 Para mostrar um diff do que foi feito no commit:  
`# git log -p`
- 4 Para mostrar o número de itens que foram removidos e inseridos no commit:  
`# git log --stat`

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Formatando saída

1 Uma forma especial do git log, mostra os commits por colaborador do projeto:  
# git shortlog

2 Para mostrar uma representação ASCII do repositório:  
# git log --graph

3 O --graph pode ser combinado a outros parâmetros:  
# git log --graph --oneline --decorate

4 Para formatar a saída do git log com dados diferentes, podemos usar o --pretty:  
# git log --pretty=format:"%an %ae %P"

Uma lista completa com os possíveis parâmetros, pode ser encontrada no link:  
[https://www.kernel.org/pub/software/scm/git/docs/git-log.html#\\_pretty\\_formats](https://www.kernel.org/pub/software/scm/git/docs/git-log.html#_pretty_formats)

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Filtrando a saída

Veremos agora como filtrar as informações do **git log**:

1 Filtragem por quantidade:

```
# git log -2
```

2 Filtragem por data:

```
# git log --after="2017-7-1"
```

```
# git log --before="2017-7-1"
```

3 Filtragem por autor:

```
# git log --author="Gabriel"
```

```
# git log --committer="2017-7-1"
```

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Filtrando a saída

1 Filtrar pela mensagem de commit:  
# git log --grep="BVO-123"

2 Filtrar por arquivo:  
# git log -- run.py project/settings.py

3 Filtrar por conteúdo inserido ou removido em algum commit:  
# git log -S"Hello"

4 Remover os commits com merge da listagem:  
# git log --no-merges

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Criando Alias

O Git permite a configuração de alias para executar outros comandos, por exemplo para criar um alias para o comando `git log --decorate --oneline --graph`, executamos:

1 Comando log:

```
# git log --decorate --oneline --graph
```

2 Comando log:

```
# git config alias.lg "log --decorate --oneline --graph"
```

3 Comando log:

```
# git lg
```

Pode-se utilizar as opções `--global`/`--local`/`--system` para especificar onde será salvo o alias

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Outros comandos

**git show** mostra informações mais detalhadas sobre um commit e diversos tipos de objetos.

- 1 Mostra o commit atual com um diff do que foi realizado:

```
# git show
```

- 2 Parecido com o anterior, deixa as informações dos commits resumidas:

```
# git show --pretty=oneline
```

- 3 Exibe apenas o nome dos arquivos modificados e a informação do commit:

```
# git show --name-only
```

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Outros comandos

**git grep** faz uma busca no conteúdo dos arquivos e retorna a linha com a palavra ou frase especificada.

- 1 Retorna os arquivos que possuem uma linha com a palavra Hello:

```
# git grep Hello
```

- 2 Retorna os arquivos e o número da linha com a palavra Hello:

```
# git grep --line-number Hello
```

- 3 Permite que insira uma regex para buscar conteúdos específicos:

```
# git grep --perl-regex '[he].'
```

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---



# GitLab

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Objetivos da Aula

- ✓ O que é.
- ✓ Versões.
- ✓ Instalação.
- ✓ Funcionalidades.
- ✓ Backup.



## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Conceito

- ✓ Gitlab é uma aplicação web para gerenciar repositórios privados.
- ✓ Possui a mesma função do Github, porém você pode instalar em sua infraestrutura e gerenciar o serviço internamente.
- ✓ Desenvolvido em Ruby.



## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Característica do Gitlab



Interface web.



Gerenciamento de repositórios  
Git: atividades, wikis, reviews.



Web Hooks.



Escalável (25.000 usuários  
em cluster ativo).

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

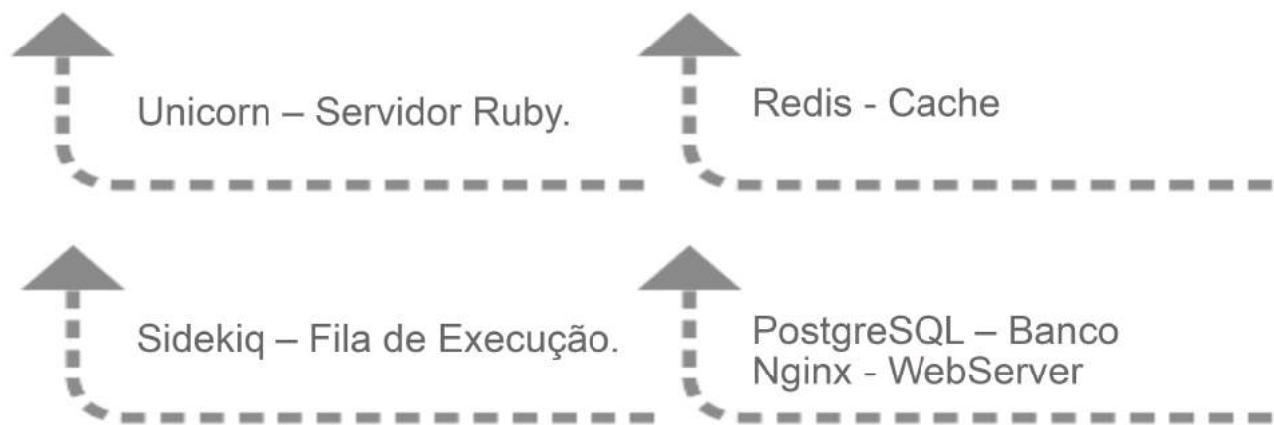
---

---

---

---

### Componentes do Gitlab



### Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Versões

- ✓ CE - Community Open Source.
- ✓ EE – Enterprise.

O Gitlab pode rodar como local ou hospedado pela Gitlab.com, nas duas versões CE e EE.

A principal diferença, consiste o suporte oferecido de acordo com o plano, podendo ter features extras como sincronismo com múltiplos LDAPs/ADs, suporte 24/7, recuperação de desastre, testes de segurança, Canary Deployments entre outros.

<https://about.gitlab.com/pricing/#self-hosted>

## Quem utiliza



<https://about.gitlab.com/case-studies/360i-transitions-to-gitlab/>

Empresa de mídia, detém clientes como Toyota, Oreo, Ben & Jerry's.

Gostaria de migrar de SVN para GIT.

Dados precisavam ficar in-loco.

Pedido dos desenvolvedores + Problemas com repositórios de clientes

Gitlab foi testado, e customizado.

Após seguir para produção verificou-se melhoria na colaboração.

## Instalação

1 Execute:

```
# sudo apt-get install curl openssh-server ca-certificates postfix
```

2

```
# curl -sS  
https://packages.gitlab.com/install/repositories/gitlab/gitlab-  
ce/script.deb.sh | sudo bash
```

3

```
# sudo apt-get install gitlab-ce
```

4

```
# sudo gitlab-ctl reconfigure
```

Ao executar o comando gitlab-ctl, reconfigure o deamon do gitlab que irá reler as configurações dentro do arquivo /etc/gitlab/gitlab.rb. Neste arquivo podemos definir o endereço de acesso da ferramenta, portas e quantidade de memória utilizada.

A versão que está sendo instalada nesse caso é a *Community Edition*, que já possui todos os serviços pré instalados como: Nginx, Radius, PostgreSQL, Prometheus, Sidekiq, Unicorn entre outros.

The screenshot shows the GitLab Community Edition password creation interface. At the top, there is a header with the 4Linux logo and the word "Gitlab". Below the header, the title "Introdução a Interface" is displayed. A small orange fox icon is centered above a blue bar. The blue bar contains the text "Please create a password for your new account.". To the right, there is a "Change your password" form with fields for "New password" and "Confirm new password", and a "Change your password" button. Below the form, links for "Didn't receive confirmation instructions?" and "Already have login and password? Sign in" are provided. At the bottom left, there are links for "Explore", "Help", and "About GitLab".

Esse primeiro acesso irá definir a senha do usuário root do Gitlab, sendo esse admin do sistema.

## Introdução a Interface

Após definir a senha, vamos logar no Gitlab.

[Sign in](#)      [Register](#)

Username or email

Password

Remember me      [Forgot your password?](#)

[Sign in](#)

Didn't receive a confirmation email? [Request a new one.](#)

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

The screenshot shows the GitLab interface. At the top, there is a navigation bar with links for GitLab, Projects, Groups, Activity, Milestones, Snippets, and a search bar. Below the navigation bar, the main content area is titled "Welcome to GitLab" with the subtitle "Code, test, and deploy together". There are four main sections: "Create a project" (with a brief description and icon), "Create a group" (with a brief description and icon), "Add people" (with a brief description and icon), and "Configure GitLab" (with a brief description and icon). The background of the main content area has a subtle grid pattern.

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Gerenciando Usuários

Vamos adicionar um novo usuário na área de administração do Gitlab.

The screenshot shows the GitLab Admin Area Dashboard. At the top, there is a navigation bar with links for GitLab, Projects, Groups, Activity, Milestones, Snippets, and a search bar. Below the navigation bar, the dashboard has three main sections: 'Projects: 1' with a 'New project' button, 'Users: 2' with a 'New user' button, and 'Groups: 0' with a 'New group' button. On the left side, there is a sidebar with various icons for managing projects, users, and groups.

### Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

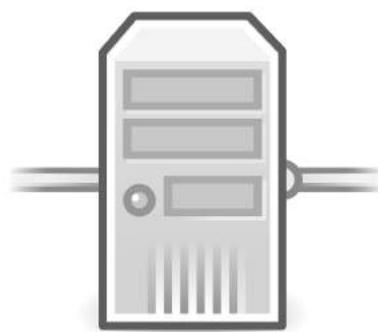
---

---

---

## Repositórios remotos

Tudo o que realizamos no treinamento até o momento, foi executado localmente. Entretanto, é importante compreendermos, como as mudanças produzidas podem ser disponibilizadas para mais pessoas.



- ✓ git pull
- ✓ git remote
- ✓ git fetch
- ✓ git push

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## git remote

Gerencia um ou mais repositórios remotos.

- 1 Para adicionar um repositório remoto ao projeto:

```
# git remote add origin <enderço do repo>
```

- 2 Listar os repositórios remotos:

```
# git remote -v
```

- 3 Alterar o endereço do repositório remoto já adicionado:

```
# git remote set-url origin <novo endereço>
```

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## git fetch

Obtém os objetos novos de um repositório remoto.

- 1 Para trazer os novos objetos da branch master, do remoto, para a branch master local:  
`# git fetch origin master`

- 2 Para trazer todos os objetos:  
`# git fetch --all`

Diferente da opção pull, o comando git fetch realizará o download dos novos objetos de um repositório, mas sem efetuar o merge com o repositório local, esse passo precisa ser feito manualmente com a opção git merge reporemoto/branch.

## git pull

Obtém e faz o merge dos commits, de um repositório remoto a um branch local.

1

Trazer os novos objetos e efetuar o merge das mudanças:  
# git pull origin master



O comando irá buscar as diferenças do repositório chamado origin e mesclá-las ao branch no qual estamos atualmente.

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## git push

Envia o que foi executado no repositório local para o repositório remoto.

1

Para enviar os commits da branch autal para a branch master do repositório remoto:  
# git push origin master



O Git push envia todos os commits que estão a frente do repositório remoto.  
Para subir as mudanças, seu repositório local deve estar com os commits mais recentes do repositório remoto.

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Backup

**1**

Realize o backup:

```
# gitlab-rake gitlab:backup:create
```

**2**

```
# ls /var/opt/gitlab/backups/
```

```
1462942348_gitlab_backup.tar 1462942804_gitlab_backup.tar
```

**3**

Efetue o restore:

```
# gitlab-rake gitlab:backup:restore BACKUP=1462942804
```

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Funcionalidades

✓ MERGE REQUEST

✓ WEBHOOKS

✓ PROTECTED BRANCHES

✓ API

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Funcionalidades – Merge Request

Merge Request permite ao desenvolvedor solicitar ao administrador do projeto no Gitlab que seja feito um merge do código para outra branch. A opção abaixo aparecerá quando o desenvolvedor realizar uma ação de push para o repositório em uma branch diferente da master.

Curso CI/CD 4Linux > dexter > **Details**

You pushed to **dev** at **Curso CI/CD 4Linux / dexter** 30 seconds ago

**Create merge request**

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Funcionalidades – Merge Request

Curso CI/CD 4Linux > dexter > **Merge Requests**

### New Merge Request

From dev into master

[Change branches](#)

Title

Dev

Start the title with `WIP:` to prevent a **Work In Progress** merge request from being merged before it's ready.

Add [description templates](#) to help your contributors communicate effectively!

Description

[Write](#) [Preview](#)

Write a comment or drag your files here...



Markdown and [quick actions](#) are supported

[Attach a file](#)

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

The screenshot shows a GitLab Merge Request interface. At the top, there's a header with the 4Linux logo and the word "Gitlab". On the right, it says "108". Below the header, the title "Funcionalidades – Merge Request" is displayed. Underneath the title, the path "Curso CI/CD 4Linux > dexter > Merge Requests > !2" is shown. A green "Open" button indicates the request is active. It was opened 27 seconds ago by "Curso CI/CD 4Linux". There are "Edit" and "Close merge request" buttons. The main content area is titled "Update README.md" and contains the message "Novo README.md file". Below this, a "Request to merge dev into master (1 commit behind)" section is shown. It includes a "Merge" button (which is checked), a "Remove source branch" checkbox, and a "Modify commit message" button. A note says "You can merge this merge request manually using the command line". At the bottom of this section are three icons: thumbs up (0), thumbs down (0), and a smiley face.

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

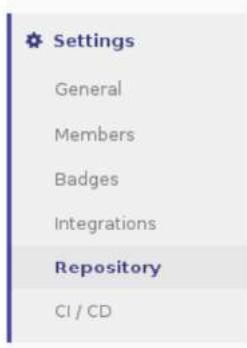
---

---

---

## Funcionalidades – Protected Branches

Dentro do Repositório, accese o menu Settings > Repository.



### Protected Branches

Keep stable branches secure and force developers to use merge requests.

[Collapse](#)

By default, protected branches are designed to:

- prevent their creation, if not already created, from everybody except Maintainers
- prevent pushes from everybody except Maintainers
- prevent anyone from force pushing to the branch
- prevent anyone from deleting the branch

[Read more about protected branches and project permissions.](#)

A screenshot of a 'Protect a branch' dialog box. It has a light gray header with the title 'Protect a branch'. Below the header, there are three dropdown menus:

- 'Branch:' dropdown set to 'Select branch or create wildcard'. A note below it says 'Wildcards such as \*-stable or production/\* are supported'.
- 'Allowed to merge:' dropdown set to 'Maintainers'.
- 'Allowed to push:' dropdown set to 'Developers + Maintainers'.

At the bottom of the dialog box is a single button labeled 'Protect'.

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Funcionalidade - WEBHOOKS

Dentro do repositório de um projeto acesse Settings > Integrations para adicionar um Webhook.

The screenshot shows the 'Integrations' section of the GitLab settings interface. On the left, a sidebar lists 'General', 'Members', 'Badges', 'Integrations' (which is selected and highlighted in grey), 'Repository', and 'CI / CD'. The main area has a heading 'Integrations' with a sub-instruction: 'Webhooks can be used for binding events when something is happening within the project.' Below this are fields for 'URL' (containing 'http://example.com/trigger-ci.json'), 'Secret Token' (an empty field), and a note: 'Use this token to validate received payloads. It will be sent with the request in the X-Gitlab-Token HTTP header.' Under the 'Trigger' section, there are three checkboxes: 'Push events' (checked, with a note: 'This URL will be triggered by a push to the repository'), 'Tag push events' (unchecked, with a note: 'This URL will be triggered when a new tag is pushed to the repository'), and 'Comments' (unchecked, with a note: 'This URL will be triggered when someone adds a comment')).

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Funcionalidades - API

- ✓ Possibilita integração com qualquer ferramenta.
- ✓ Permite realizar listagem das informações do repositório.
- ✓ Viabiliza que alterações sejam feitas no repositório.

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

**Personal Access Tokens**

You can generate a personal access token for each application you use that needs access to the GitLab API.

You can also use personal access tokens to authenticate against Git over HTTP. They are the only accepted password when you have Two-Factor Authentication (2FA) enabled.

**Add a personal access token**

Pick a name for the application, and we'll give you a unique personal access token.

**Name**

**Expires at**

**Scopes**

- api** Access the authenticated user's API
- read\_user** Read the authenticated user's personal information
- sudo** Perform API actions as any user in the system (if the authenticated user is an admin)
- read\_repository** Read Repository

Read Repository

**Create personal access token**

## Gitlab – API Token

Para criarmos um Token de uso da API devemos acessar Profile Settings > Access Tokens, definir um nome, data de expiração e scope. O scope pode ser:

Api Access your API, (Acesso completo a API, podendo ler e gravar informações em todos os endpoints).

read\_user Read user information (Acesso de leitura no endpoint /users).

read\_registry Read Registry (Acesso de leitura no registro de imagens).

Para mais detalhes sobre a API do Gitlab: <https://docs.gitlab.com/ce/api/>

## API

Your new personal access token has been created.

### Personal Access Tokens

You can generate a personal access token for each application you use that needs access to the GitLab API.

You can also use personal access tokens to authenticate against Git over HTTP. They are the only accepted password when you have Two-Factor Authentication (2FA) enabled.

#### Your New Personal Access Token

tCs8xjg7omkGmrxu\_a

Make sure you save it - you won't be able to access it again.

#### Add a personal access token

Pick a name for the application, and we'll give you a unique personal access token.

Name

Expires at

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## API — GET /api/v4/users

```
[  
  {  
    "id": 1,  
    "username": "john_smith",  
    "name": "John Smith",  
    "state": "active",  
    "avatar_url":  
      "http://localhost:3000/uploads/user/avatar/1/cd8.jpeg",  
      "web_url": "http://localhost:3000/john_smith"  
  },  
  {  
    "id": 2,  
    "username": "jack_smith",  
    "name": "Jack Smith",  
    "state": "blocked",  
    "avatar_url": "http://gravatar.com/.../e32131cd8.jpeg",  
    "web_url": "http://localhost:3000/jack_smith"  
  }  
]
```

Para realizar a consulta acima, utilize o Token gerado anteriormente por exemplo, com comando curl:

```
curl --header "Private-Token: <TOKEN>" http://<endereço gitlab>/api/v4/users | json_pp
```

## API — GET /api/v4/users?username=devops

```
[  
  {  
    "name": "devops",  
    "username": "devops",  
    "id": 2,  
    "state": "active",  
    "avatar_url": "http://www.gravatar.com/avatar/  
85f00568bc37a00fc685fd2908361d2d?s=80&d=identicon",  
    "web_url": "http://gitlab.example.com/u/devops"  
  }  
]
```

### Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Workflows

Abordar fluxos de desenvolvimento no Git, não é algo simples. A sua forma flexível de trabalho, permite que diversos desses fluxos sejam criados, atendendo a necessidades específicas de cada projeto.



- ✓ Centralized Workflow.
- ✓ Integration-manager Workflow.
- ✓ Git Flow.

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

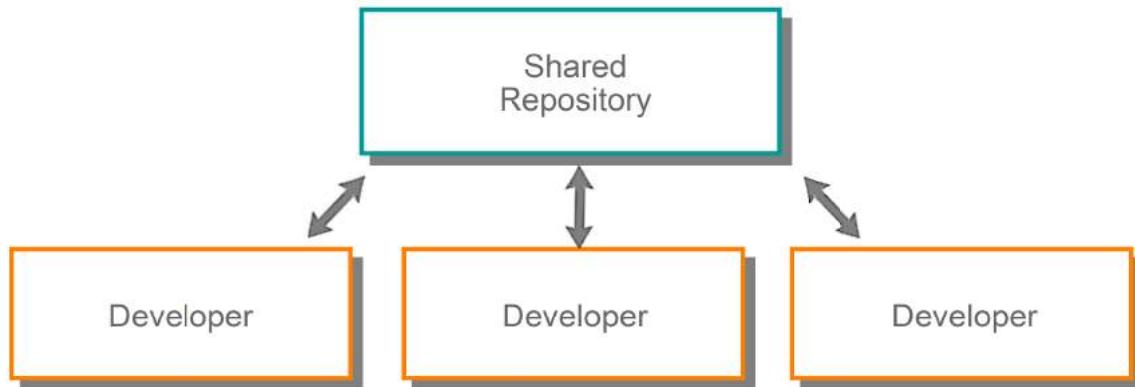
---

---

---

## Centralized Workflow

Esse é um fluxo simples para trabalhar com Git. Normalmente usado por equipes pequenas, ou para projetos com poucos commits.



[https://git-scm.com/book/en/v2/images/centralized\\_workflow.png](https://git-scm.com/book/en/v2/images/centralized_workflow.png)

## Centralized Workflow

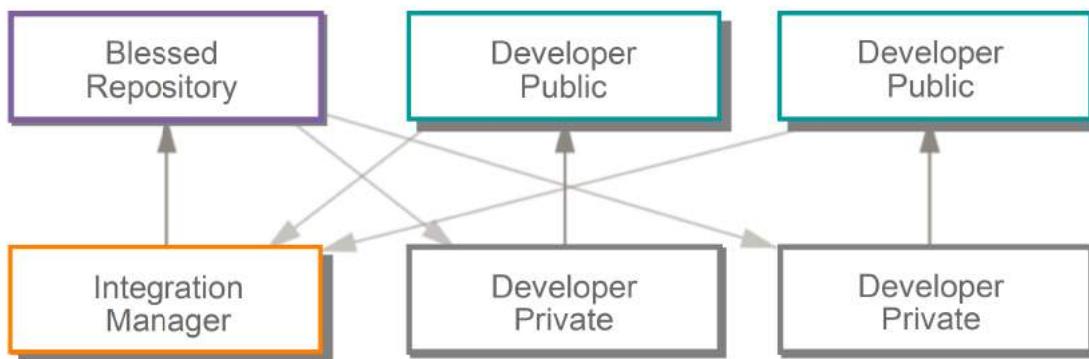
Neste fluxo, o primeiro desenvolvedor que realizar o push para o repositório remoto não terá problemas, já o segundo precisará fazer o merge com as alterações submetidas pelo primeiro.

Esse workflow pode ser implementado criando um único repositório e, fornecendo somente acesso a ação de push. Assim que o segundo desenvolvedor submeter o código, um erro no servidor irá recusar, informando que um fetch e merge é necessário.

Ideal para grupos pequenos.

## Integration-Manager Workflow

Neste workflow, cada desenvolvedor possui seu próprio repositório e realiza suas modificações individualmente. Ao término, avisa um responsável que, caso aprovado, encaminhará ao repositório oficial.



<https://git-scm.com/book/en/v2/images/integration-manager.png>

## Integration-Manager Workflow

O gerente do projeto realiza o push para o repositório público.

O contribuidor clona o repositório e realiza suas alterações locais.

O contribuidor faz o push para seu próprio repositório público.

O contribuidor informa o gerente, solicitando um pull de sua cópia.

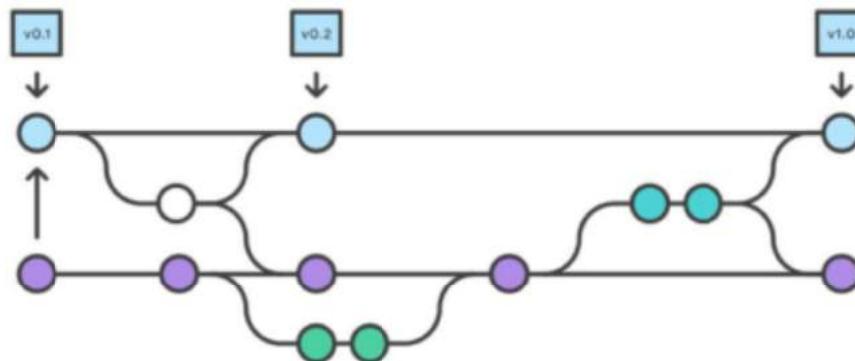
O gerente adiciona o repositório solicitado e faz o merge local para teste.

Caso aprovado o gerente faz o push para o repositório principal.

Essa implementação é recomendada para equipes maiores, desmembradas, onde os projetos possam estar em ferramentas como Github e Gitlab.

## Git Flow

Este fluxo de trabalho, definido por uma estrutura de branches que interagem em momentos específicos, é uma excelente forma de gerenciar projetos de grande porte.



<https://www.atlassian.com/dam/jcr:e3bd4199-27ac-4bac-a5d2-3ff0fdb112d3/01.svg>

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

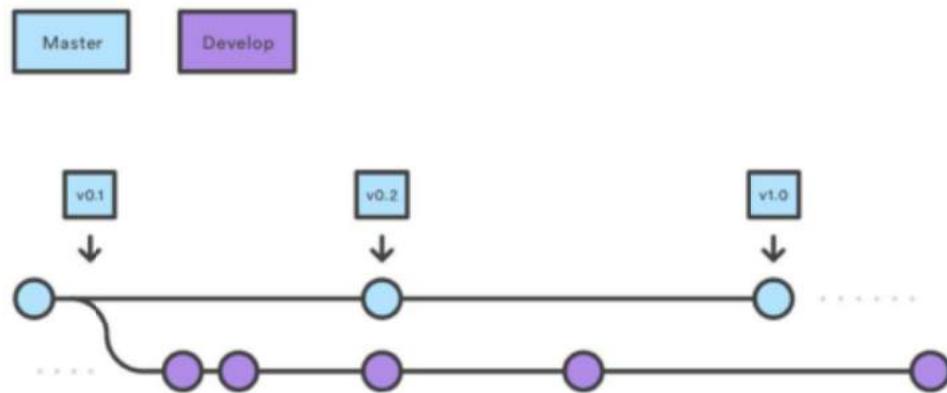
---

---

---

## Branches master e dev

Este fluxo de trabalho usará uma branch master e outra dev, para guardar o histórico de mudanças e interagir com as outras branches.



<https://www.atlassian.com/dam/jcr:2e2315b0-d79a-403f-a981-4cb94599df1f/02.svg>

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

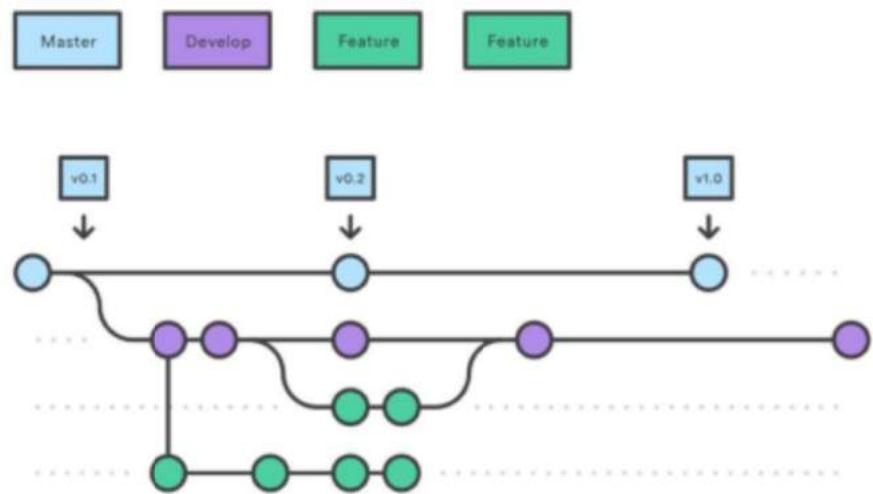
---

---

---

## Branch feature

Cada nova **feature** do projeto deverá possuir uma branch separada, sendo todas inicialmente, uma cópia da branch dev.



<https://www.atlassian.com/dam/jcr:9f149cef-f784-43de-8207-3e7968789a1f/03.svg>

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

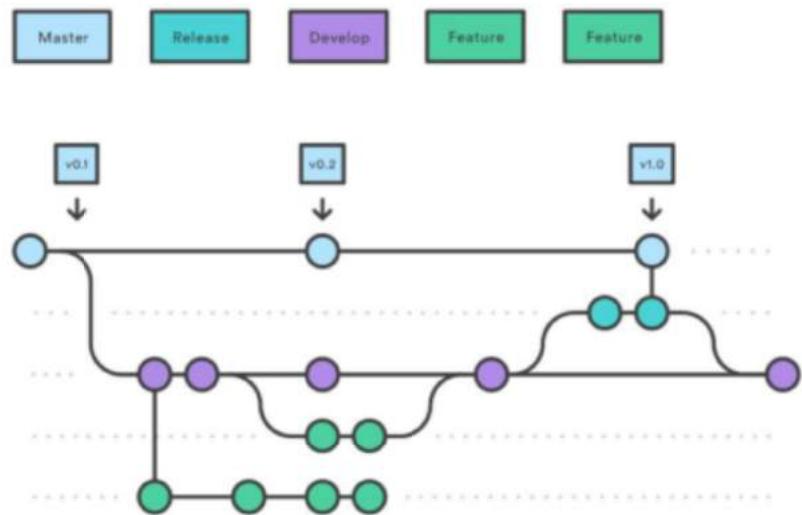
---

---

---

## Branch release

Depois de todas as features serem desenvolvidas e adicionadas a branch dev, uma branch de release será criada, virará a branch candidata a nova versão.



<https://www.atlassian.com/dam/jcr:3555a856-675e-453a-b49d-ba60667809e1/04.svg>

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

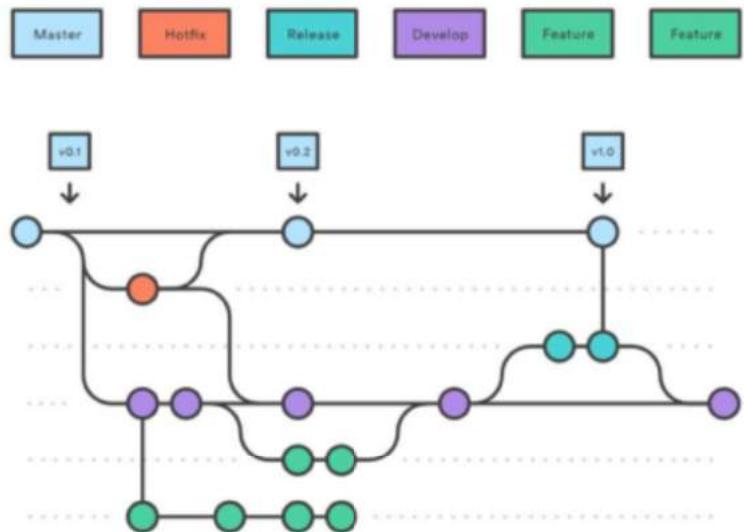
---

---

---

## Branch hotfix

A última branch é a de hotfix, que serve para situações emergenciais.



<https://www.atlassian.com/dam/jcr:21cf772d-2ba5-4686-8259-fcd6fd2311df/05.svg>

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---



# Jenkins

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Objetivos da Aula

- ✓ O que é Jenkins.
- ✓ Vantagens.
- ✓ Linguagens e plataformas suportadas.
- ✓ Boas práticas.
- ✓ Pré-requisitos.
- ✓ Instalação
- ✓ Primeiro Acesso
- ✓ Instalação de Plugins
- ✓ Jobs e Projects
- ✓ Integração com Gitlab
- ✓ Dependência entre Jobs/Projects

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

**Jenkins** é uma aplicação para Continuous Integration (CI) e Continuous Delivery (CD). Possui integração com várias ferramentas, inclusive com o Gitlab.



## Continuous Integration

Continuous Integration ou Integração Contínua, constitui uma prática de desenvolvimento na qual os membros de um time, integram seu trabalho frequentemente, normalmente essa integração é realizada diariamente. Cada integração é verificada por uma tarefa ou build automatizada, que detecta erros de integração da forma mais rápida possível. A integração contínua diminui a probabilidade de erros dentro de um projeto.

## Continuous Delivery

Continuous Delivery ou Entrega Contínua, consiste uma prática de desenvolvimento usada na construção de software, de maneira que mudanças possam ser disponibilizadas no ambiente de produção a qualquer momento. Em um ambiente de entrega contínua, são realizadas mudanças pequenas e com maior frequência de Deploys, o que diminui o risco de erros em ambientes de produção, simplesmente pelo fato de serem pequenas mudanças.

## Características do Jenkins



Interface web.



Multiplataforma  
(Desenvolvido em Java)



Integração com a maioria  
dos SCM's.

Pode ser instalado via docker, native system packages ou Standalone, em qualquer contêiner web Java.

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Linguagens suportadas

- ✓ PHP
- ✓ .Net
- ✓ Ruby
- ✓ Scala
- ✓ Python
- ✓ node.js
- ✓ Perl
- ✓ C++
- ✓ etc

## Plataformas suportadas

- ✓ Debian/Ubuntu/Mint ...
- ✓ Red Hat/Fedora/CentOS
- ✓ Windows
- ✓ Mac OS
- ✓ Solaris
- ✓ Gentoo
- ✓ Solaris
- ✓ Outras...

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Vantagens

- ✓ Identifica erros mais cedo, a cada commit.
- ✓ Publica os resultados e envia emails de notificação.
- ✓ Flexível: bash scripts, shell scripts, ANT, Maven, etc...
- ✓ Reduz trabalho.
- ✓ Facilita a entrega, deploy contínuo.
- ✓ Gera documentação.
- ✓ Integração com outras ferramentas: Sonar, Nexus...
- ✓ EVITA O GO HORSE PROCESS!

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Vantagens

- 1- Pensou. Não é XGH.
- 2- Três formas para resolver um problema: a correta, a errada e a XGH, igual a errada, porém mais rápida.
- 3- Quanto mais XGH feito, mais precisará fazer.
- 4- XGH é totalmente reativo.
- 5- XGH vale tudo, só não vale...
- 6- Commit sempre antes de update.
- 7- XGH não tem prazo.
- 8- Prepare-se para pular fora quando o barco começar a afundar... Ou coloque a culpa em alguém ou algo.
- 9- Seja autêntico, XGH não respeita padrões.
- 10- Não existe refactoring, apenas rework



- 1- Pensou. Não é XGH. XGH não pensa, faz a primeira coisa que vem a mente. Não existe segunda opção, a única opção é a mais rápida.
- 2- Três formas para resolver um problema: a correta, a errada e a XGH, igual a errada, porém mais rápida. XGH é mais rápido que qualquer metodologia de desenvolvimento de software que você conhece.
- 3- Quanto mais XGH feito, mais precisará fazer. Para cada problema resolvido, mais 7 são criados, XGH tende ao infinito.
- 4- XGH é totalmente reativo. Os erros só existem quando aparecem.
- 5- XGH vale tudo, só não vale... Resolveu o problema? Compilou? Commit, era isso.
- 6- Commit sempre antes de update. Se der merda, a sua parte estará sempre correta. Seus colegas que se...
- 7- XGH não tem prazo. Os prazos vindos dos clientes, são meros detalhes. Você sempre conseguirá implementar tudo no tempo necessário, mesmo que implique acessar o BD por um script malaco.

8- Prepare-se para pular fora quando o barco começar a afundar... Ou coloque a culpa em alguém ou algo. Para adeptos do XGH, um dia o barco afunda. Quanto mais o tempo passa, o sistema vira um monstro. O dia que a casa cair, melhor seu currículo estar ativo, ou repasse a culpa.

9- Seja autêntico, XGH não respeita padrões. Escreva o código como você bem entender, se resolver o problema, commit e era isso.

10- Não existe refactoring, apenas rework. Se der merda, refaça um XHG rápido que solucione o problema. O dia que o rework implicar em reescrever a aplicação toda, pule fora, o barco irá afundar.

## Boas práticas

Build a cada commit.

- ✓ Build deve executar todos os testes da aplicação.
- ✓ Mantenha o tempo de build curto.
- ✓ Pelo menos uma Job para cada ambiente.
- ✓ Commits constantes\*.
- ✓ Tornar fácil o acesso ao artefato do último deploy.
- ✓ Automatize o deploy.
- ✓ O SCV deve acionar a build no Jenkins.



## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Pré-requisito

Iniciamos com a instalação do Java 8.

1 # sudo add-apt-repository ppa:webupd8team/java

2 # sudo apt-get update

3 # sudo apt-get install oracle-java8-installer

## Requisitos de instalação do Jenkins

### Java

```
# sudo add-apt-repository ppa:webupd8team/java  
# sudo apt-get update  
# sudo apt-get install oracle-java8-installer
```

### Instalando Jenkins

#### Método 1

Download Jenkins: <http://mirrors.jenkins.io/war-stable/latest/jenkins.war>  
# java -jar jenkins.war

#### Método 2

```
# wget -q -O - | sudo apt-key add -  
  
# sudo sh -c 'echo deb binary/ > /etc/apt/sources.list.d/jenkins.list'  
  
# sudo apt-get update  
  
# sudo apt-get install jenkins
```

## Instalando Jenkins

### Método 3

Download Jenkins: <http://mirrors.jenkins.io/war-stable/latest/jenkins.war>

```
# cp jenkins.war wildfly/standalone/deployments
```

### Método 4

Download Jenkins: <http://mirrors.jenkins.io/war-stable/latest/jenkins.war>

```
# cp jenkins.war $tomcat/webapps
```

## Instalação Jenkins – Iniciando Serviço

1 Iniciar Jenkins:

```
# sudo systemctl start jenkins
```

2 Verificar status:

```
# sudo systemctl status jenkins
```

3 Deve aparecer a mensagem similar:

```
jenkins.service - LSB: Start Jenkins at boot time
Loaded: loaded (/etc/init.d/jenkins; bad; vendor preset:
enabled)
Active: active (exited) since Thu 2017-04-20 16:51:13
UTC; 2min 7s ago
Docs: man:systemd-sysv-generator(8)
```

## Iniciando Jenkins

Após instalar o Jenkins por qualquer um dos métodos anteriormente apresentados, execute os comandos que seguem para inicia-lo:

```
# sudo systemctl start jenkins
```

```
# sudo systemctl status jenkins
```

## Instalação Jenkins – Primeiro Acesso

# Unlock Jenkins

To ensure Jenkins is securely set up by the administrator, a password has been written to the log ([not sure where to find it?](#)) and this file on the server:

`/var/jenkins_home/secrets/initialAdminPassword`

Please copy the password from either location and paste it below.

Administrator password

Após seguir um dos passos de instalação do Jenkins, e inicia-lo, teremos uma tela no primeiro acesso ao endereço `http://localhost:8080`, semelhante a apresentada no slide. Nesta, o Jenkins solicita que copiemos a senha inicial de administrador, localizada no arquivo `<jenkins_home>/secrets/initialAdminPassword`.

# Customize Jenkins

Plugins extend Jenkins with additional features to support many different requirements.

## Install suggested plugins

Install plugins the Jenkins community finds most useful.

## Select plugins to install

Select and install plugins most suitable for your needs.

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

✓ Folders Plugin	✓ OWASP Markup Formatter Plugin	✓ build timeout plugin	✓ Credentials Binding Plugin
✓ Timestamper	✓ Workspace Cleanup Plugin	✓ Ant Plugin	✓ Gradle Plugin
✓ Pipeline	✓ GitHub Branch Source Plugin	✓ Pipeline: GitHub Groovy Libraries	✓ Pipeline: Stage View Plugin
✓ Git plugin	✗ Subversion Plug-in	✗ SSH Slaves plugin	✓ Matrix Authorization Strategy Plugin
✓ PAM Authentication plugin	✓ LDAP Plugin	✗ Email Extension Plugin	✓ Mailer Plugin

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Página inicial

The screenshot shows the Jenkins home page. At the top right, it says "Bem-vindo ao Jenkins!". Below that, a message says "Por favor clique em: [Novo job](#) para iniciar.". On the left, there's a sidebar with links: "Novo job", "Usuários", "Histórico de compilações", "Gerenciar Jenkins", and "Credentials". Below the sidebar, there are two sections: "Fila de builds" (which is empty) and "Estado do executor de builds" (which shows 1 Parado and 2 Parado).

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Plugins

O Jenkins é um software extensível, significa que podemos facilmente adicionar novas funcionalidades através de plugins. Qualquer um pode desenvolver plugins para o Jenkins, o que o torna, uma ferramenta com grandes possibilidades de configurações e alterações.



## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Gerenciar Jenkins

Clique no botão **Gerenciar Jenkins**, para entrar nas páginas de configurações do Jenkins.



## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Gerenciar Jenkins

Procure a opção Gerenciar Plugins.



	<u>Configurar o sistema</u> Configurar opções globais e caminhos
	<u>Configurar segurança global</u> Faça a segurança no Jenkins; defina quem pode usar/acessar o sistema.
	<u>Recarregar configuração do disco</u> Descartar todos os dados carregados na memória e recarregar tudo do sistema de arquivos. Isso é útil quando seus arquivos de configuração foram modificados diretamente no disco.
	<u>Gerenciar plugins</u> Adiciona, remove, desabilita e habilita plugins que podem incrementar as funcionalidades do Jenkins. ( <b>Atualizações disponíveis</b> )
	<u>Informações do sistema</u> Mostrar várias informações do ambiente para auxiliar na resolução de problemas.
	<u>Log do sistema</u> O log do sistema captura a saída de java.util.logging relacionada ao Jenkins.

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Gerenciar Jenkins

Selecione a aba **Disponíveis** para instalar os plugins.



Instalar ↓	Nome
<input type="checkbox"/>	<a href="#"><u>.NET Development</u></a>
<input type="checkbox"/>	<a href="#"><u>CCM Plug-in</u></a> This plug-in generates reports on cyclomatic complexity
<input type="checkbox"/>	<a href="#"><u>FxCop Runner plugin</u></a> <u>FxCopCmd.exe</u> execute plugin.
	<a href="#"><u>MSBuild Plugin</u></a>

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

# 4LINUX / Plugins do Jenkins

144

## Gerenciar Jenkins

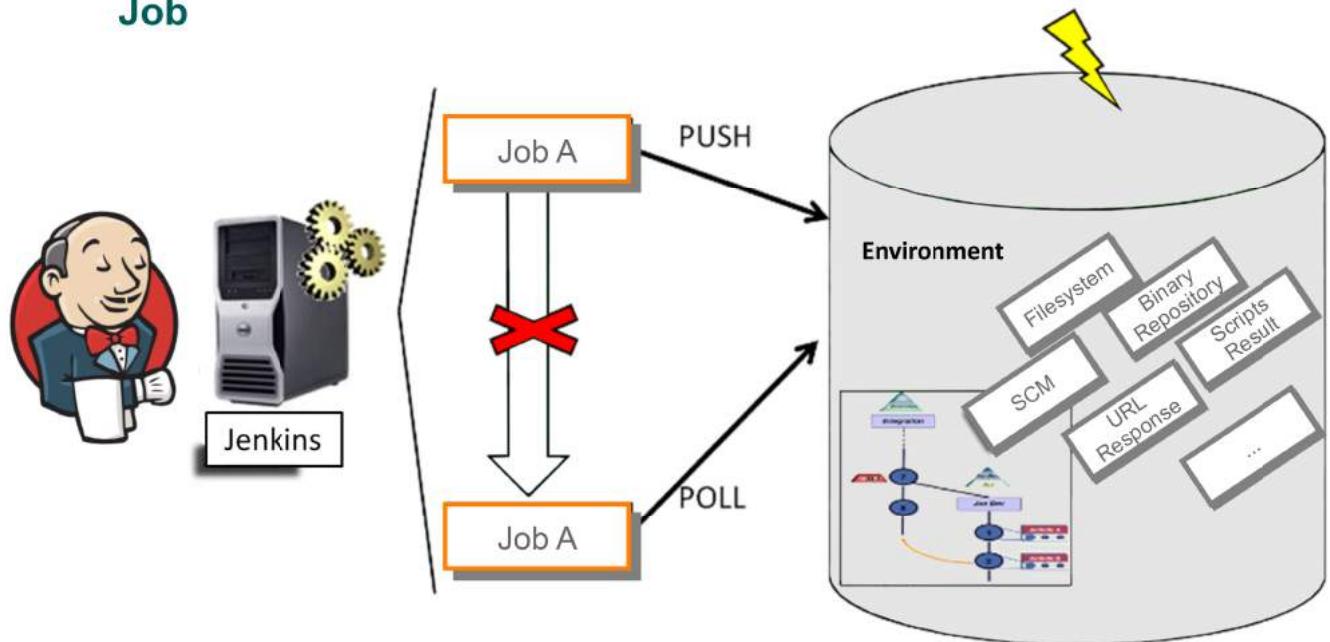
Após selecionar todos os plugins, clique no botão **Baixar Agora** e aguarde a instalação.



Durante o curso iremos utilizar alguns plugins para integração com ferramentas como Gitlab e SonarQube, como:

- Gitlab
- Sonar Quality Gates Plugin
- SonarQube Scanner for Jenkins
- Build Pipeline

## Job



Para o Jenkins, tudo acontece dentro de uma job. Simplificando o termo, podemos pensar que uma job é uma tarefa ou, passo a ser executado no processo de build de uma determinada aplicação. Uma aplicação pode ter uma ou mais jobs integrando seu ambiente de integração contínua.

Uma job geralmente é responsável pelo processo de compilação, executa testes automatizados (unitários, integração...), deploy. Também é capaz de invocar outra job de acordo com seu status (sucesso ou falha).

## Job

Apesar do termo ser muito utilizado, é considerado depreciativo. O termo atualmente utilizado é 'Project'.

### Definição do Jenkins para 'Project'

Uma descrição de trabalho definida pelo usuário, que o Jenkins executará, como a construção de um software, etc.

Todos os termos utilizados pelo Jenkins podem ser encontrados em uma glossário:  
<https://jenkins.io/doc/book/glossary/>

## Job

Apesar do termo job ser o mais utilizado, é considerado depreciativo.

A expressão mais atual seria: "project".

Jenkins possui uma página com todos os termos utilizados dentro de sua plataforma.

A definição do Jenkins para project:

***Uma descrição de trabalho configurada pelo usuário, que o Jenkins deve executar, como a construção de um software, etc.***

<https://jenkins.io/doc/book/glossary/>

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Tipos de Jobs

- ✓ Projetos de software Free-Style.
- ✓ Construir um Projeto de Múltipla Configuração.
- ✓ External Jobs.

Projetos de software Free-Style: são projetos para um propósito geral, que disponibiliza maior flexibilidade de ferramentas e tarefas no Jenkins.

Projeto de Múltipla Configuração: define um projeto que pode ser executado para várias outras configurações. Normalmente é utilizado para ambientes de teste de uma aplicação em ambientes diferentes, com banco de dados diferentes e máquinas diferentes.

External Jobs: usado para monitorar a execução de processos não interativos, como tarefas da crontab do sistema e procmail.

## Criar um Job no Jenkins.

**Jobs ou Projects** são tarefas gerenciadas e monitoradas pelo Jenkins.



The screenshot shows the Jenkins dashboard. At the top, there's a navigation bar with the Jenkins logo and a dropdown menu. Below it is a sidebar with links: 'Novo job' (New Job), 'Usuários' (Users), 'Histórico de compilações' (Build History), 'Gerenciar Jenkins' (Manage Jenkins), and 'Credentials'. The main content area is titled 'Fila de builds' (Build Queue) and displays the message 'Nenhum build na fila.' (No builds in the queue).

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

The screenshot shows the Jenkins dashboard. On the left, there's a sidebar with icons for 'Novo job', 'Usuários', 'Histórico de compilações', 'Gerenciar Jenkins', and 'Credentials'. A green hand icon points from the 'Novo job' button towards a teal arrow pointing right. This arrow points to a modal window titled 'Enter an item name'. The window contains a text input field labeled 'Nome da Job' with a note '(Required field)'. Below the input field is a list of item types with descriptions:

- Construir um projeto de software free-style**: Esta é a central de funcionalidades do Jenkins. Ele construirá seu projeto,e você pode combinar qualquer SCM com qualquer sistema de software.
- Pipeline**: Orchestrates long-running activities that can span multiple build slaves. Suitable for building pipelines (formerly known as workflows).
- External Job**: Este tipo de trabalho permite que você grave a execução de um processo rodando fora do Jenkins (talvez até de uma máquina remota) dentro de seus sistemas de automação.
- Construir projeto de múltiplas configurações**: Apropriado para projetos que necessitam de grande número de diferentes configurações, como teste em múltiplos ambientes, builds de múltiplos pacotes, etc.
- Folder**: Creates a container that stores nested items in it. Useful for grouping things together. Unlike view, which is just a filter, a folder creates a physical container.
- Github Organization**: Scans a GitHub organization (or user account) for all repositories matching some defined markers.
- Multibranch Pipeline**: Creates a set of Pipeline projects according to detected branches in one SCM repository.

If you want to create a new item from other existing, you can use this option:

Copy from  Type to autocomplete

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

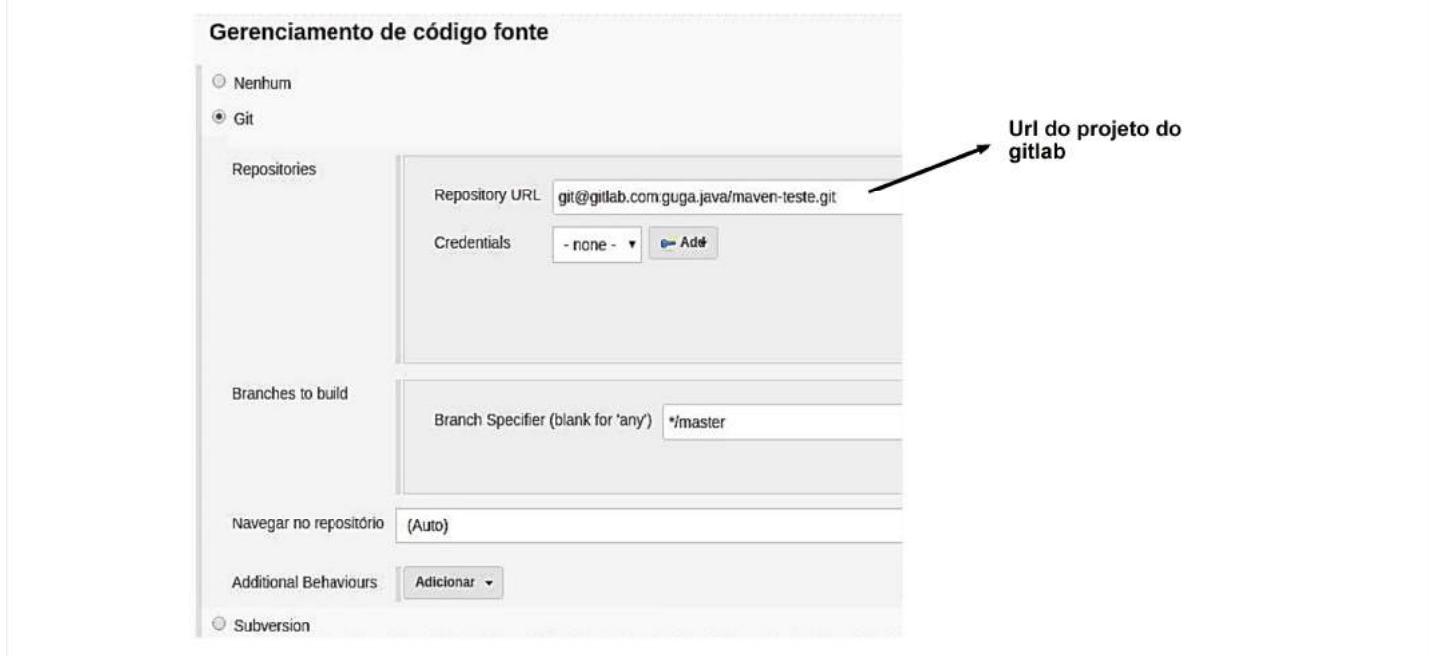
---

---

---

# 4LINUX / Criando o projeto

150



Selecionar opção git

Definir endereço do repositório

Caso a conexão seja realizada utilizando SSH, será necessário criar uma credencial ou, adicionar a chave privada ao diretório .ssh da home do usuário jenkins em /var/lib/jenkins.

Definir branch que o project irá sincronizar, no exemplo utilizamos o branch master.

## Criar um Job no Jenkins.

### Trigger de builds

- Dispare builds remotamente (exemplo, a partir dos scripts)
- Construir após a construção de outros projetos
- Construir periodicamente

Agenda

*****	
-------	--

Selecionar: "construir periodicamente"

Definir: "\*\*\*\*\*"

Dessa forma o Jenkins, a cada minuto, verificará se houve alguma alteração na branch do repositório Git informado. Essa não é a melhor maneira para verificar as mudanças no código fonte. Mostraremos uma maneira mais eficiente para fazer isso!

## Criar um Job no Jenkins.

Na seção de Build, definimos todos os passos de compilação, execução de testes automatizados, deploy, etc.

O Jenkins é bem versátil nessa etapa, com possibilidade de executar comandos maven, ant, gradle, utilizar shell script, entre outros.

The screenshot shows the Jenkins configuration interface for a job. The main title is 'Build'. Below it is a dropdown menu labeled 'Adicionar passo no build'. A list of build steps is visible, including: 'Chamar alvos Maven de alto nível', 'Executar no comando do Windows', 'Executar shell', 'Invocar Ant', 'Invoke Gradle script', 'Run with timeout', and 'Set build status to "pending" on GitHub commit'. At the bottom of the window are two buttons: 'Salvar' (Save) and 'Apply'.

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

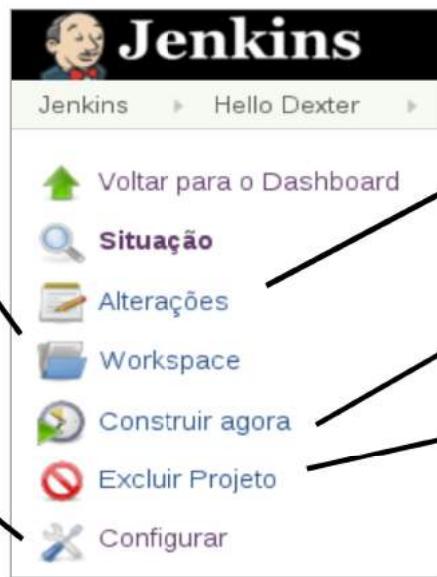
---

---

## Menu do Projeto

Arquivos criados pelas Builds.

Reconfigurar o projeto.



Histórico de mudanças.

Executar uma tarefa manualmente.

Deletar um projeto.

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Tarefa

Vamos executar manualmente a nossa primeira Tarefa no Jenkins. Clique no botão **Construir Agora** para executar o projeto.



## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

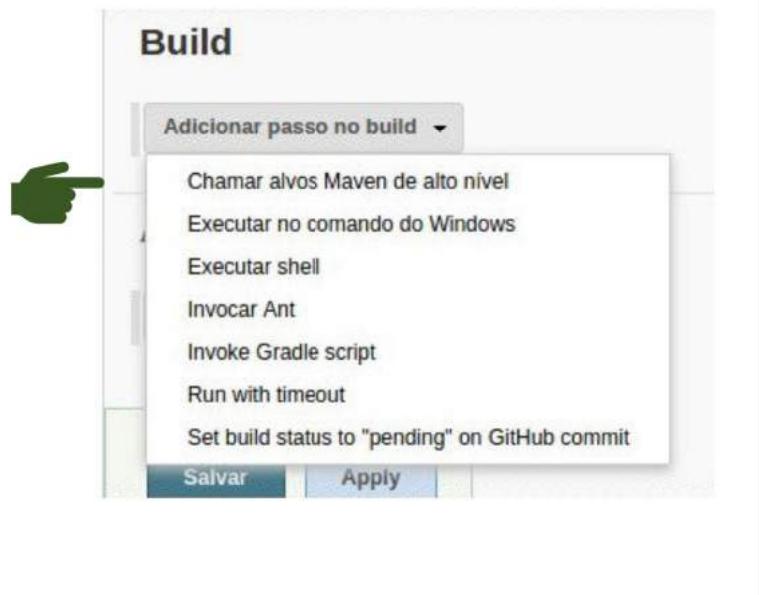
---

---

---

## Tarefa

Selecionar a opção “Chamar alvos Maven de alto nível”. Para o nosso exemplo vamos utilizar os comandos “clean package” do maven.



## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Histórico de builds

The screenshot shows a Jenkins interface titled "Histórico de builds". At the top, there's a search bar labeled "find" and a link to "Tendência". Below the search bar, a single build entry is listed: "#1" (with a blue circular icon), followed by the build timestamp "23/02/2016 15:21". At the bottom of the list, there are two RSS feed links: "RSS para todos" and "RSS por falhas".

Podemos visualizar o histórico de Builds do projeto.  
Uma Build representa a construção de um Job do Jenkins.  
É possível visualizar a saída dos comandos utilizados,  
basicamente um log das Builds.

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Log da Build

Na opção **Saída do Console** podemos acompanhar o resultado dos comandos executados pelo Jenkins. Clique no link para visualizar qual foi a saída da Build em questão.

The screenshot shows the Jenkins web interface. At the top, there's a header with the Jenkins logo and the text "Jenkins" followed by a breadcrumb trail: "Hello Dexter" > "#1". Below the header is a navigation menu with the following items:

- Voltar ao projeto (Back to project)
- Estado pessoal (Personal status)
- Alterações (Changes)
- Saída do console** (Console output) - This item is highlighted with a green arrow pointing to it.
- Editar informações de compilação (Edit compilation information)
- Excluir este build (Delete this build)

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Indicadores de estabilidade

ÍCONES	Porcentagem de builds construídas com sucesso
	80-100% das builds foram construídas com sucesso.
	60-79% das builds foram construídas com sucesso.
	40-59% das builds foram construídas com sucesso.
	20-39% das builds foram construídas com sucesso.
	0-19% das builds foram construídas com sucesso.

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Integração com Gitlab

Vamos configurar a nossa Job Hello Dexter para ser executada toda vez que alguém modificar o repositório do Gitlab.

Assim, teremos o Gitlab disparando uma trigger para o Jenkins, avisando que houve mudanças no repositório.



## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Instalando Plugin GitLab

Gerenciar Jenkin >>> Gerenciar Plugins >>> Disponíveis  
Pesquise por **Gitlab Plugin**.

Nome
<a href="#">Gitlab Authentication plugin</a> A Jenkins authentication plugin that delegates to gitlab. We also implement an Authorization Strategy that users the acquired OAuth token Jenkins.
<a href="#">GitLab Logo Plugin</a> Display GitLab Repository Icon on dashboard
<a href="#">Gitlab Merge Request Builder</a> A plugin to build merge requests in Gitlab
<a href="#">Violation Comments to GitLab Plugin</a> Finds violations reported by code analyzers and comments GitLab merge requests with them.
<a href="#">Gitlab Hook Plugin</a> Enables Gitlab web hooks to be used to trigger SMC polling on Gitlab projects
<input checked="" type="checkbox"/> <a href="#">GitLab Plugin</a>

Atualizações   Disponíveis   Instalados   Avançado

Instalar

Nome

Instalar sem reiniciar   Baixar agora, Instalar e depois reiniciar   Update information obtained: 1 hora 8 minutos ago   Verificar agora

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Gitlab API TOKEN

- ✓ Configurações do Jenkins.
- ✓ Definir URL de acesso ao GitLab.
- ✓ Definir GitLab API Token.

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Gitlab API TOKEN

Gerenciar Jenkin >>> Manage Jenkins >>> Configure System >>> **Gitlab.**

The screenshot shows the Jenkins configuration interface for managing GitLab connections. The 'Gitlab' section is active. It includes fields for 'Connection name' (empty, with a red error message), 'Gitlab host URL' (empty, with a red error message), and 'Credentials' (a dropdown menu showing '- none -' and an 'Add' button). There are also 'Advanced...', 'Test Connection', and 'Delete' buttons.

Gitlab

Enable authentication for 'project' end-point

GitLab connections

Connection name

Gitlab connection name required.  
A name for the connection

Gitlab host URL

Gitlab host URL required.  
The complete URL to the Gitlab server (i.e. http://gitlab.org)

Credentials

API Token for Gitlab access required  
API Token for accessing Gitlab

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Gitlab API TOKEN

Para adicionar uma credencial click em Add. Defina kind como GitLab API token e em API Token utilize o Token do GitLab.



O GitLab Private token, que deverá ser copiado, está dentro de Settings do usuário, sobre o menu Account. Feita essa configuração, poderemos interligar o GitLab com o Jenkins.

## Gitlab API TOKEN

Gerenciar Jenkin >>> Manage Jenkins >>> Configure System >>> **Gitlab.**

The screenshot shows the Jenkins 'Configure System' page with the 'Gitlab' section selected. It includes fields for enabling authentication, connection name, host URL, credentials dropdown, and buttons for advanced settings, testing, and deleting the connection.

**Gitlab**

Enable authentication for '/project' end-point

GitLab connections

Connection name	A name for the connection
Gitlab	https://gitlab.com

Gitlab host URL  
The complete URL to the Gitlab server (i.e. http://gitlab.org)

Credentials

API Token for accessing Gitlab

, ,

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Configurações na Job

- ✓ Gitlab connection - Selecionar Configure dentro do Projeto no Jenkins.
- ✓ Marcar "Build when a change is pushed to GitLab".

### Trigger de builds

- Dispare builds remotamente (exemplo, a partir dos scripts)
- Build when a change is pushed to GitLab. GitLab CI Service URL: <http://jenkins-bee:8080/project/bee-front>

A URL informada, deverá ser configurada dentro do Gitlab nos próximos passos. Com isso, sempre que um evento PUSH ocorra no repositório, o Gitlab informará o Jenkins para realizar o build e os testes. Dentro de *Advanced* dessa opção terá uma campo *Secret Token*, ao precionar o botão *Generate* um hash será gerado, e esse hash deverá ser utilizado dentro do Gitlab na área de *Integration* do Projeto.

## Gitlab Web Hook

- ✓ Settings do projeto no Gitlab >> Integration
- ✓ `http://JENKINS_URL/project/PROJECT_NAME`

The screenshot shows the 'Integrations' tab selected in the top navigation bar of a GitLab project settings page. On the left, there's a sidebar with the heading 'Integrations' and a note about using Webhooks for binding events. The main area contains fields for 'URL' (set to `http://example.com/trigger-ci.json`) and 'Secret Token' (a placeholder field). Below these, under 'Trigger', there's a checked checkbox for 'Push events' with a descriptive note: 'This URL will be triggered by a push to the repository'.

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Chave SSH

Para que o Jenkins possa interagir com o servidor git remoto, no caso Gitlab, é necessário "autorizar" no Gitlab os acessos oriundos do Jenkins.

### Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Dependências entre Jobs

No Jenkins é possível configurar dependências entre as Jobs.

Em nosso projeto teremos duas Jobs: uma de Homologação e uma de Produção para cada serviço.

A job de produção é dependente da job de homologação.

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

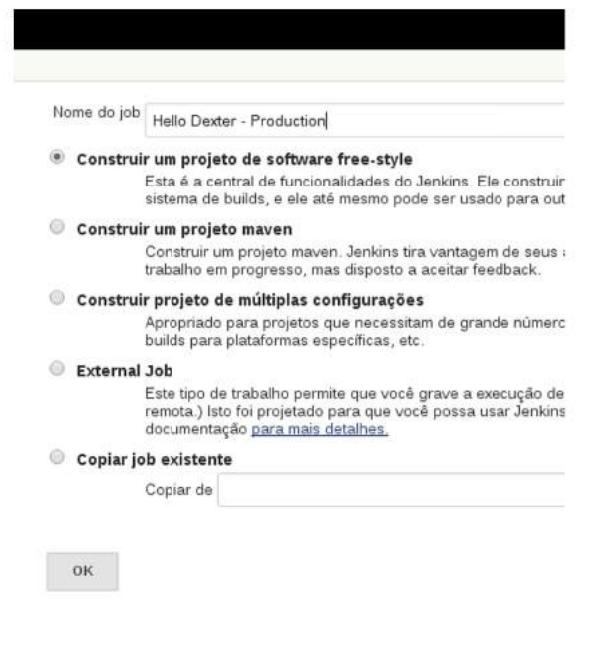
---

## Dependências entre Jobs

Vamos configurar uma segunda Job.

Nome: **Hello Dexter – Production**

Selecione a opção: **Free Style**, e clique no botão OK para criar o projeto.



## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

### Dependências entre Jobs

Volte na tela inicial do Jenkins, selecione a primeira Job criada, **Hello World**, selecione então a opção Configurar.

Jenkins > Hello Dexter > configuração

 Voltar para o Dashboard

 Situação

 Alterações

 Workspace

 Construir agora

 Excluir Projeto

 **Configurar**



### Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Dependências entre Jobs

Vamos adicionar um passo  
**Pós-Build.**  
Selecione a opção **Construir Outros Projetos.**



- Agrega o resultado dos testes com projeto filho
- Arquivar os artefatos
- Construir outros projetos
- Gravar fingerprints de arquivos para trilhar o uso
- Publicar Javadoc
- Publicar relatório de testes do JUnit
- Git Publisher
- Gitlab commit status publisher
- Build other projects (manual step)
- Notificação de E-mail
- Trigger parameterized build on other projects

Adicionar ação de pós-build ▾

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Dependências entre Jobs

Defina o nome da Build que será construída após a execução.

Clique em Salvar para finalizar a configuração.

### Construir outros projetos

Projetos para construir

Hello Dexter - Production



- Disparar apenas se o build estiver estável
- Disparar mesmo se o build estiver instável
- Disparar mesmo se o build falhar

[Excluir](#)

[Adicionar ação de pós-build ▾](#)

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Dependências entre Jobs

Selecione a Build inicial do processo: **Hello Dexter**.

Clique no botão **OK**, para finalizar a configuração da View.

This layout mode derives the pipeline structure based on the upstream/downstream trigger relationship between jobs.

Select Initial Job

Hello Dexter



1



Para testar, execute a build do projeto **Hello Dexter**.

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

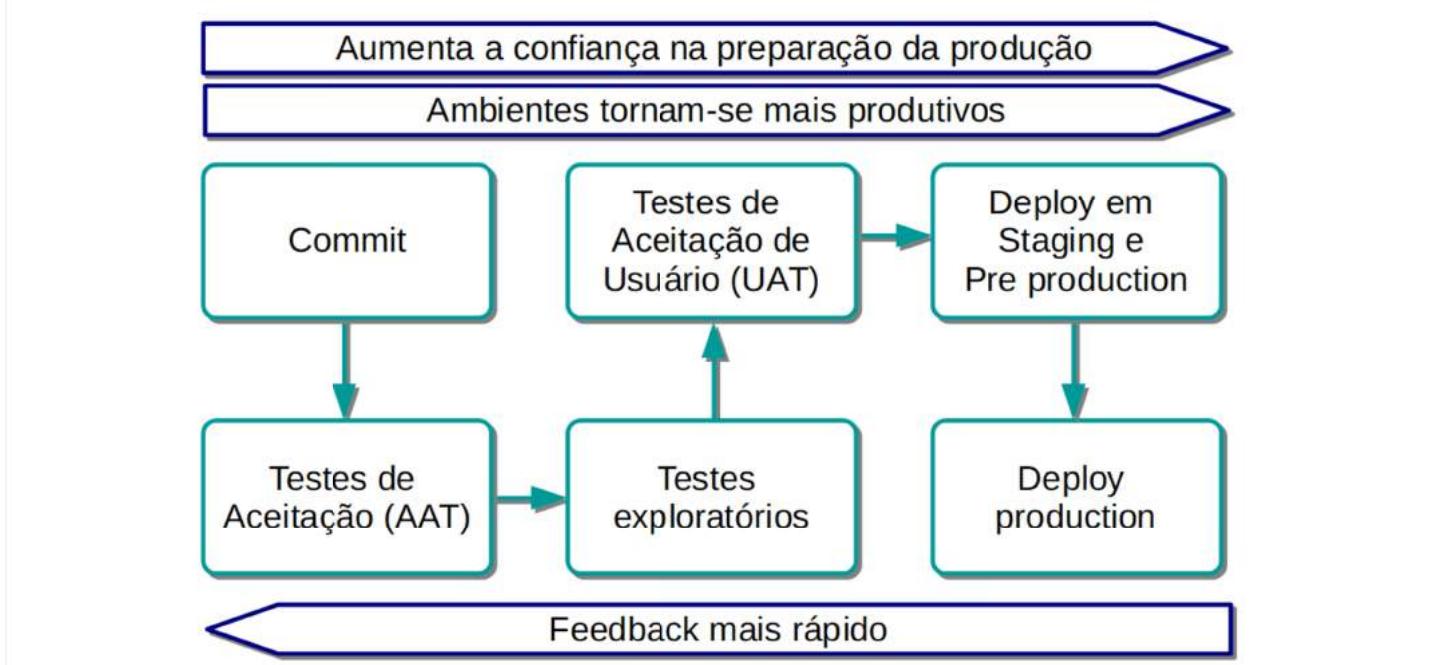
---

---

---



Referências biográficas: B5, B6, C5



A implantação (deployment) de compilações (builds) pode ter finalidade para testes como também para produção efetiva.

Em um nível abstrato, uma pipeline de implementação consiste em uma manifestação automatizada do processo de obter o software, a partir do controle de versão, até entregá-lo nas mãos dos próprios usuários.

Toda alteração no software em desenvolvimento, passa por um processo complexo de liberação e lançamento.

A etapa Commit, assegura que o sistema funciona em nível técnico. São realizadas compilações, testes automatizados, principalmente testes unitários e, análise do código.

A etapa Teste de Aceitação Automatizada ou Automated Acceptance Testing (AAT), assegura que o sistema funcione, em nível funcional e não funcional, comportando-se conforme às necessidades dos usuários e as especificações do cliente.

A etapa de testes manuais, atestam que o sistema é utilizável, cumpre seus requisitos, detecta defeitos não capturados por testes automatizados e verifica se fornece valor para seus usuários. Testes manuais podem incluir ambientes de testes exploratórios, de integração e UAT (testes de aceitação do usuário).

As etapas de Deploy entregam o sistema empacotado e pronto para usuário ou implementa em um ambiente de produção ou de teste, um ambiente de teste deve ser idêntico ao de produção.

## Dicas para criação do seu script

- ✓ Script é toda automação que suporta a compilação: teste, deploy e lançamento do software (release), e um plano de execução.
- ✓ Use os processos para build e deployment como um guia inicial, assim cada tarefa dos scripts será simplificada.
- ✓ Objetivo final: compartilhamento do mesmo mecanismo de deployment entre ambientes de desenvolvimento, testes e produção.
- ✓ Os scripts devem ser versionados, mantidos, testados e refatorados, ser o único mecanismo usado para deploy.

Usamos o termo "script" em um sentido bastante amplo. Geralmente queremos nos referir a toda a automação que nos ajuda a construir, testar, implementar e lançar nosso software. Em um primeiro olhar, a ampla coleção de scripts em um típica pipeline de implementação, pode parecer assustadoramente complexa. No entanto, cada tarefa em um script de compilação ou implementação, é simples e, o processo em si não é complexo.

É altamente recomendável usar os scripts e comandos de build e deployment existentes, como um guia inicial para criação do seu script. A partir daí, expanda-o, etapa por etapa, com a inclusão de novos recursos necessários para automatizar todo o processo de build e deployment.

Mantenha o objetivo final sempre em mente: compartilhar o mesmo mecanismo de deploy entre desenvolvimento, teste e produção. Idealmente, você deve envolver equipes de operações e desenvolvedores na criação desses mecanismos.

## Garanta fluidez

- ✓ A pipeline deve ser iniciada a cada check-in do código.
- ✓ Novas etapas devem começar imediatamente após a conclusão, bem sucedida, da anterior.
- ✓ Se você fizer, por exemplo, builds do código, a cada hora, testes de aceitação, todas as noites, e testes de carga, semanalmente, você impedirá a realização de um processo eficiente e um feedback confiável.

Finalmente, é preciso reiterar que os scripts são parte fundamental do seu sistema. Devem percorrer todo o ciclo de vida do produto. Devem ser versionados, bem como, mantidos, testados e refatorados, sendo também, o único mecanismo que você usará para implementar seu software.

É comum equipes refletirem tarde sobre seu sistema de compilação. Como resultado, esses sistemas mal gerenciados são muitas vezes a barreira para um processo de lançamento fluido e repetitivo, além da sua criação.

As equipes de entrega devem dedicar tempo e cuidado para conseguir bons scripts de criação e implementação. Gaste algum tempo, pense sobre os objetivos que deseja alcançar e projete seu processo de criação e deploy para alcançá-los

## Importante

- ✓ **Implementação Contínua:** realiza o deploy em ambiente de produção, para todas as alterações que passaram nos testes automatizados.
- ✓ **Elabore uma estratégia de lançamento:** antes mesmo de criar seu plano de execução do deployment, e da 1ª iteração.

Implantação Contínua e Entrega Contínua são muitos similares, mas é importante entender a diferença proposta. Na Entrega Contínua, o software alterado está pronto para ser entregue ao usuário, porém são liberados manualmente pelas equipes DevOps. Já na Implantação Contínua, o deployment em produção é totalmente automatizado. Os usuários recebem novas atualizações assim que criadas e devidamente testadas, sem intervenções.

À medida que o aplicativo cresce e se torna mais complexo, o mesmo acontecerá em seu pipeline de implementação. Considerando que o pipeline de implementação modele o processo de teste e de lançamento, você precisa descobrir como é esse processo. Embora o pensamento seja normalmente centrado em como fazer os builds entre ambientes, há mais detalhes para se preocupar. É importante observar:

quais etapas um build precisa percorrer para ser liberado? Por exemplo: testes de integração, teste de QA, teste de aceitação do usuário, teste de produção.

quais os requisitos necessários?

quem autoriza os builds entre cada passagem dessas etapas?

## Importante

- ✓ **Continuidade em DevOps:** nem sempre significará ausência de interrupções durante os processos.
- ✓ **Plano de iteração:** visa implementação na primeira iteração, para testá-la e provar o que pode ser feito.

É importante destacar que o termo contínuo, nem sempre consiste em implementações DevOps com fluxo totalmente sem interrupções. Apesar de todas as ferramentas de automações disponíveis, raramente o desenvolvimento do software é completamente contínuo.

Iteração é instância de uma repetição de uma ou mais ações. Da mesma forma, uma repetição é composta de uma ou mais iterações. A primeira implantação de qualquer aplicativo deve acontecer logo na primeira iteração, quando você apresenta suas histórias ou requisitos iniciais para o cliente.

Escolha uma ou duas histórias ou requisitos de alta prioridade, mas muito simples de entregar na primeira iteração, assumindo que as iterações sejam de uma ou duas semanas de duração e, você tenha uma pequena equipe, você deve escolher mais histórias se essas condições não se aplicarem.

Use este caso como argumento, para tornar o aplicativo implementável em produção, da mesma forma que no ambiente de apresentação (UAT). Um dos principais objetivos da primeira iteração de um projeto, é obter os estágios iniciais do pipeline de implementação e, ser capaz de implementar e demonstrar algo, não importa o quanto pequena ou curta seja a tarefa ao final. Essa é uma das poucas situações em que recomendamos priorizar o valor técnico sobre o valor do negócio. Você pode pensar nessa estratégia, como o pontapé inicial do seu processo de desenvolvimento.

## Dicas para adoção de implantação contínua

- ✓ As pessoas que farão a implantação, devem ser envolvidas na criação dos processos.
- ✓ Criar logs com as atividades da implantação.
- ✓ Não delete arquivos antigos, move-os.
- ✓ A implementação é responsabilidade de toda a equipe.
- ✓ Busque rapidamente falhas.
- ✓ Não faça alterações diretamente no ambiente de produção.

A objeção intuitiva à implementação contínua, consiste ser muito arriscada. Mas como observamos antes, lançamentos mais frequentes levam a menores riscos, quando de um outro lançamento maior em particular.

Você não pode fazer isso sem automatizar todo o processo de criação, deploy, teste e liberação.

Não pode realizar efetuar isso sem um conjunto abrangente e confiável de testes automatizados.

Você não pode realizar isso sem gravar testes de sistema, que sejam executados em um ambiente de produção.



## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Objetivos da Aula

- ✓ Introdução.
- ✓ Pré-requisitos e Instalação.
- ✓ Sonar como serviço;
- ✓ Integração com Maven.
- ✓ Integração com Jenkins.

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

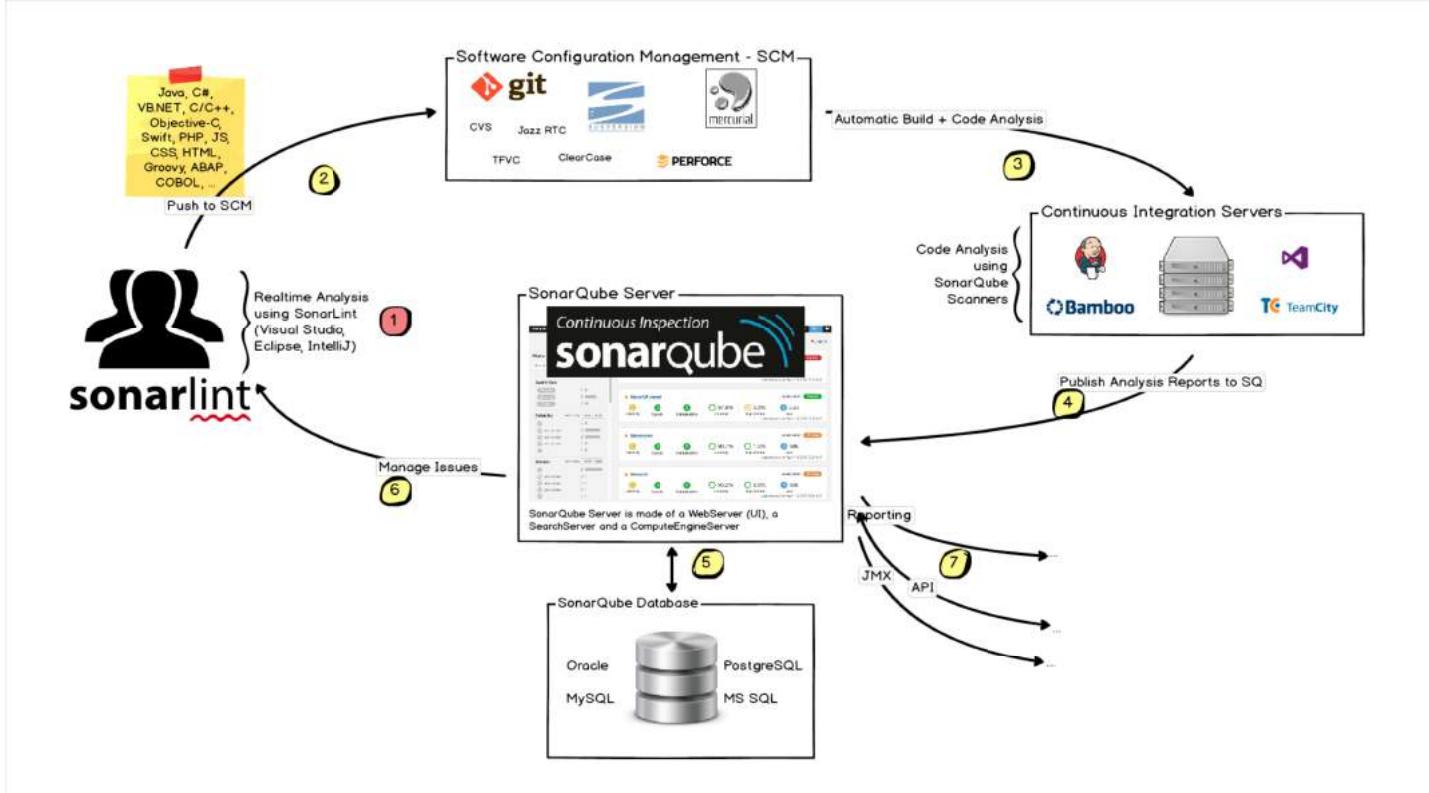
---

---

---

---

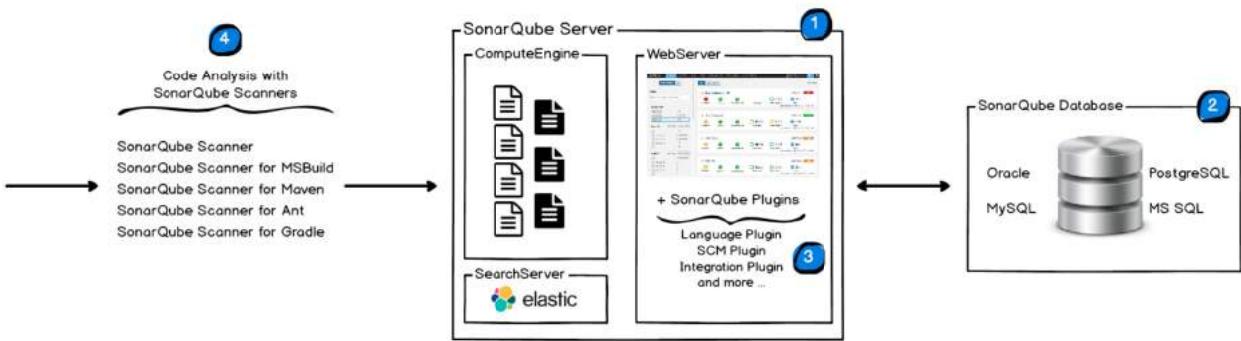
---



## SonarQube Workflow

- 1 – Desenvolvedores utilizando o SonarLint integrado com as ferramentas de desenvolvimento Eclipse, Visual Studio, IntelliJ, etc.
- 2 – Push para uma solução de SCM como Git e Gitlab.
- 3 – O SCM ativará uma solução de CI como Jenkins ou Bamboo, que rodará não somente os testes mas também redirecionará para o SonarQube Server analisar o código submetido.
- 4 – Push para o SonarQube Server realizar o Code Analysis.
- 5 – SonarQube irá registrar as informações de análise.
- 6 – O desenvolvedor realizará as melhorias e correções baseado na análise de código do SonarQube Server.
- 7 – Relatórios, APIs de comunicação.

## Introdução



## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Introdução

**Continuous Inspection** é um processo desenhado, para tornar a qualidade do código, parte integrante do ciclo de vida do desenvolvimento de software.

Mas, o que garante, ou, o que torna significativa a qualidade de código?



### Continuous Inspection

Trata-se de um novo paradigma da qualidade de software, projetado para ser parte integral do ciclo de desenvolvimento. Sendo um processo holístico e completamente realista, melhora tanto a qualidade do projeto quanto sua visibilidade para os investidores e gerentes.

Continuous Inspection fornece um gerenciamento da qualidade de código e, aumenta drasticamente o ROI (Return Of Investment) do projeto. O principal conceito por trás de Continuous Inspection consiste na identificação de problemas rapidamente enquanto sua correção contínua é barata e simples.

Sobre este modelo, auditorias são realizadas diariamente e disponibilizadas dentro da organização. Auditorias automatizadas analisam o código do projeto em múltiplas vertentes de mantinibilidade, testa se existem bugs, e compara ao padrão de codificação da empresa. O processo de auditoria é completado por ferramentas que detectam esses problemas em ambiente de desenvolvimento, similar ao corretor ortográfico de um Microsoft Word ou Libre Office.

Os membros do projeto são alertados assim que um novo problema é identificado, permitindo aos mesmos realizar as correções rapidamente, enquanto o código está na produção. A rápida identificação, e notificação, treina os desenvolvedores na adoção de hábitos melhores de codificação.

## Continuous inspection

Características da qualidade de software:

1. Capaz de executar todas as suas funcionalidades com base em critérios de aceitação.
2. Atender a todos os requisitos não funcionais: escalabilidade, confiabilidade, portabilidade, desempenho, entre outros.
3. O código deve ter o mínimo de débito técnico possível.

Qualidade de software, também pode ser medida em, quanto rápido, um desenvolvedor consegue agregar regra de negócio e valor ao software.

### Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Débito técnico

O termo foi originalmente criado por Ward Cunningham.

*“Entregar código imaturo é assumir um débito.*

*Um pequeno débito que acelera o desenvolvimento, contanto que seja pago, assim que possível, com uma revisão de código fonte.*

*A orientação a objetos torna este custo tolerável. O perigo ocorre quando o débito não é pago.*

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Débito técnico

*Cada minuto gasto com um código ruim, conta como juros da dívida.*

*Organizações de engenharia de software inteiras, podem ser levadas a um “beco sem saída”, sob o peso do débito de uma implementação ruim, orientada a objetos ou não.*

**Ward Cunningham, 1992, The WyCash Portfolio Management System,**  
**tradução de Cleuton Sampaio.**

### Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

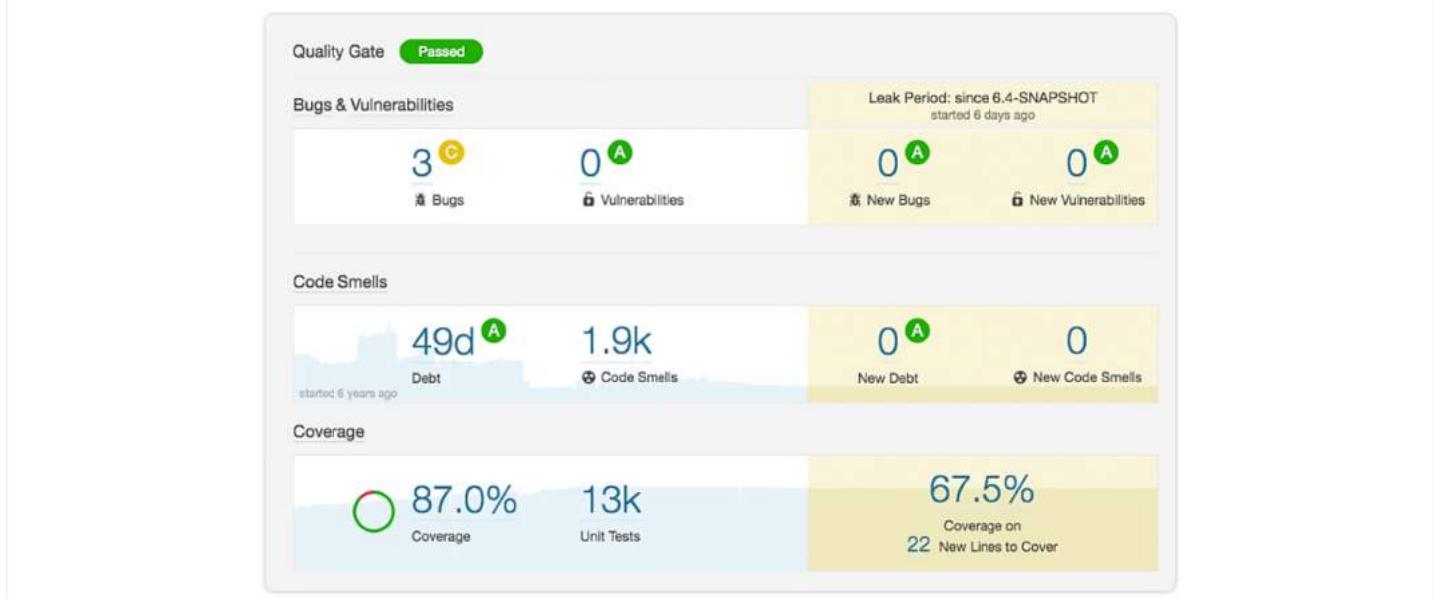
---

---

---

---

## Code smells



## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Code smells

O Sonar utiliza o conceito de code smells, como parte do processo de avaliação da qualidade de software.

*Mas o que é code smells?*

Refere-se a qualquer sintoma no código-fonte, de um programa que possivelmente indique um problema mais profundo.

Que fique claro, **code smells não são bugs**.

Nem funcionalidades implementadas de forma incorreta.

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

São códigos com design fraco, impactam negativamente no software, diminuindo a produtividade e aumentando o risco de bugs, erros e falhas no futuro. Code smells, pode ser um grande indicador de technical debt (débito técnico).

Provavelmente, o code smell mais fácil de identificar e corrigir, que acontece com maior frequência, é a duplicação de código. Códigos idênticos e espalhados pela codificação, tornam-se ineficientes em longo prazo.

Quanto mais códigos duplicados e espalhados, maiores serão os pontos de atenção e potenciais bugs. Os problemas poderão acontecer, na medida que a aplicação for atualizada, já que a atualização do código duplicado ocorrerá em diversas partes do sistema.

Large class: classes muito grandes.

Feature envy: classe que usa métodos de outra classe em excesso.

Inappropriate intimacy: classe com dependências ou detalhes de implementação de outra classe.

Lazy class / freeloader: classe que não faz quase nada.

Excessive use of literals: abusar dos literais, principalmente em configurações de sistema onde os valores deveriam estar externalizados, em arquivos de configuração.

Cyclomatic complexity: muitos desvios condicionais ou loops, podem indicar que o código deva ser quebrado em funções menores.

Too many parameters: uma longa lista de parâmetros, é difícil de ler e dificulta sua chamada.

Long method: método, função ou procedure muito extenso.

Excessive return of data: função ou método que retornam, mais que os invocadores precisam.

## Issues

Ao executar uma análise, o Sonar cria um issue, sempre que um pedaço de código quebra uma regra de codificação. O conjunto de regras de codificação, é definido através do perfil de qualidade (**Quality Profiles**) associado ao projeto.

Cada issue possui um nível de severidade: **Blocker, Critical, Major, Minor,**

**Info.**

app/app.py

Remove the unused local variable "info". 



Code Smell ▾ Minor ▾ Open ▾ Not assigned ▾ 5min effort Comment

1. Blocker: bug com uma alta probabilidade de impactar o comportamento da aplicação em produção: vazamento de memória, conexão JDBC não fechada. O código DEVE ser imediatamente corrigido.
2. Critical: bug com baixa probabilidade de impactar o comportamento do aplicativo em produção, também representa um problema de uma falha de segurança: bloco de captura vazio, injeção de SQL. O código DEVE ser revisado imediatamente.
3. Major: baixa qualidade que pode afetar a produtividade do desenvolvedor: pedaço de código descoberto, blocos duplicados, parâmetros não utilizados.
4. Minor: baixa qualidade, que pode afetar ligeiramente a produtividade do desenvolvedor: as linhas não devem ser muito longas, as declarações de "switch" devem ter pelo menos 3 casos.
5. Info: nem um erro, nem uma falha de qualidade, apenas uma constatação.

## Issues

Em um cenário ideal, os desenvolvedores não devem introduzir novas issues no projeto, para isso o Sonar fornece plugins para as principais IDE's, de modo que o desenvolvedor consiga receber a inspeção de código localmente: **SonarLint for Eclipse**, **SonarLint for IntelliJ**, **SonarLint for VisualStudio**.

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Instalação



Faça o **Download** e descompacte e execute o script de instalação.

### Download

```
cd /opt/ && wget  
https://sonarsource.bintray.com/Distribution/sonarqube/sonarqub  
e-X.Y.zip
```

Em outros sistemas operacionais, execute:

```
/opt/sonarqube-X.Y/bin/[OS]/sonar.sh console
```

## Requisitos de Hardware

1 - O SonarQube necessita de no mínimo 2GB de RAM para rodar eficientemente e, 1GB livre para o OS.

2 – A quantidade de espaço em disco necessária dependerá da quantidade de código que será analisado pelo SonarQube. Como exemplo, o SonarCloud, versão cloud do SonarQube, possui mais de 30 milhões de linhas de código em análise em mais de 4 anos de história. SonarCloud roda em uma instância EC2 m4.large, na Amazon, usando cerca de 10Gb de espaço em disco. Lida com mais de 800 projetos, com mais de 3 milhões de Issues abertas. A base de dados está consumindo 15Gb, rodando sobre a versão 9.5 do PostgreSQL.

3 – SonarQube precisa ser instalado em um hd com excelente performance de leitura e escrita, impactando o desempenho geral do SonarQube.

## Requisitos de Software

### Java

Oracle JRE – 8

OpenJDK – 8

### Database

PostgreSQL – 8.x e 9.x (UTF8 charset)

MSSQL Server - 2014 e 2016 com Microsoft JDBC driver.

Oracle – 11G, 12C e XE com UTF8.

MySQL – 5.6 e 5.7 com suporte UTF8.

The screenshot shows the SonarQube interface with the following details:

- Project Summary:** aliflix (Passed)
  - Bugs: 1 (E)
  - Vulnerabilities: 1 (B)
  - Code Smells: 4 (A)
  - Coverage: -
  - Duplications: 0.0%
- Last analysis: July 31, 2018, 4:45 PM
- Language: Python (113)
- Project Summary:** Dexter (Passed)
  - Bugs: 0 (A)
  - Vulnerabilities: 1 (B)
  - Code Smells: 10 (A)
  - Coverage: 0.0%
  - Duplications: 0.0%
- Last analysis: July 18, 2018, 9:09 PM
- Language: Java (208)

**Projects** – Resumo dos projetos analisados dentro do Sonar

**Issues** – Problemas identificados nos códigos, como Bugs, Code Smells e Vulnerabilities.

**Rules** – Regras de Codificação

**Quality Profiles** – Grupo de Rules

**Quality Gates** – Filtros de Definição

**Administration** – Área de Administração do Sonar

A Administrator

Profile Security Notifications Projects

Tokens

If you want to enforce security by not providing credentials of a real SonarQube user to run your code scan or to invoke web services, you can provide a User Token as a replacement of the user login. This will increase the security of your installation by not letting your analysis user's password going through your network.

Name	Created
SonarToken	July 16, 2018

Generate New Token:  Generate

Revoke

Dentro do profile do usuário deveremos configurar um Token para autenticação com os plugins SonarScanner e Sonar Quality Gates do Jenkins.

**IMPORTANTE:** O hash do token será apresentado somente uma única vez, tente salvar em bloco de notas.

## Integração com Jenkins

1. O primeiro passo é instalar o plugin do Sonar.  
Acesse o Jenkins, depois clique em, "Gerenciar Jenkins", em seguida "Gerenciar plugins".
2. Pesquise por SonarQube Scanner for Jenkins.
3. Adicione também o Sonar Quality Gates Plugin

Instale os plugins  
e reinicie o Jenkins



## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Integração com Jenkins

Vamos configurar o endereço do servidor Sonar.

Clique em "**Gerenciar Jenkins**", depois "**Global Tool Configuration**"

Clique em "**Adicionar SonarQube Scanner**" e defina um nome para a instalação.

### SonarQube Scanner

SonarQube Scanner instalações

Adicionar SonarQube Scanner

Lista de SonarQube Scanner instalações nesse sistema

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Integração com Jenkins

Vamos configurar o endereço do servidor Sonar.

Clique em "**Gerenciar Jenkins**", depois "**Configure System**"

No item "**SonarQube servers**" defina um Nome, Endereço do SonarQube e Chave de autenticação.

Name	Sonar Server
Server URL	http://192.168.56.3:9000
Default is http://localhost:9000	
Server authentication token	*****
SonarQube authentication token. Mandatory when anonymous access is disabled.	

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Integração com Jenkins

Em Build clique em:

"Execute SonnarQube Scanner".

Agora definiremos duas propriedades obrigatórias em nossas configurações, o identificador único para a aplicação, e a pasta raiz, onde nosso código está armazenado.

### Build

The screenshot shows a Jenkins build configuration window. At the top, there's a section for 'Goals' with 'clean deploy' selected. Below this is a dropdown menu labeled 'Adicionar passo no build'. A list of options is shown, with 'Execute SonnarQube Scanner' highlighted with a blue background. Other options include 'Chamar alvos Maven de alto nível', 'Executar no comando do Windows', 'Executar shell', 'Invoke Ant', 'Invoke Gradle script', 'Run with timeout', 'Set build status to "pending" on GitHub commit', 'SonarQube Scanner for MSBuild - Begin Analysis', and 'SonarQube Scanner for MSBuild - End Analysis'.

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Integração com Jenkins

Execute SonarQube Scanner	
Task to run	<input type="text"/>
JDK	(Inherit From Job) JDK to be used for this SonarQube analysis
Path to project properties	<input type="text"/>
Analysis properties	<pre>sonar.projectKey=aliflix sonar.sources=app sonar.projectName=Aliflix sonar.projectVersion=\${BUILD_ID}</pre>
Additional arguments	<input type="text"/>
JVM Options	<input type="text"/>

**sonar.projectKey** = Identificação do Projeto.

**sonar.sources** = Local onde o source está armazenado.

**sonar.projectName** = Nome do projeto na dashboard do Sonar.

**sonar.projectVersion** = Versão do código dentro do Sonar.

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Integração Sonar Quality Gates com Jenkins

Em **Post-build** clique em: "**Quality Gates Sonarqube Plugin**".

Defina o **Project Key** com o mesmo nome definido na etapa de **Build**.

Depois defina a ação a ser tomada quando o Sonar informar que a aplicação não passou na análise.



## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Integração com Jenkins

Project aliflix



### SonarQube Quality Gate

aliflix OK  
server-side processing: Success

### Permalinks

- [Last build \(#25\). 7 min 24 sec ago](#)
- [Last stable build \(#25\). 7 min 24 sec ago](#)
- [Last successful build \(#25\). 7 min 24 sec ago](#)
- [Last failed build \(#24\). 11 min ago](#)
- [Last unsuccessful build \(#24\). 11 min ago](#)
- [Last completed build \(#25\). 7 min 24 sec ago](#)

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

Sonatype Nexus

Sonatype Nexus é um gerenciador de repositórios

## Objetivos da Aula

- ✓ Porque usar.
- ✓ Instalação.
- ✓ Linguagens e plataformas suportadas.
- ✓ Boas práticas.
- ✓ Integração contínua.

### Armazenamento:

Manter todas as dependências de sua aplicação com uma fonte confiável. Podendo armazenar bibliotecas criadas pela Equipe de desenvolvimento.

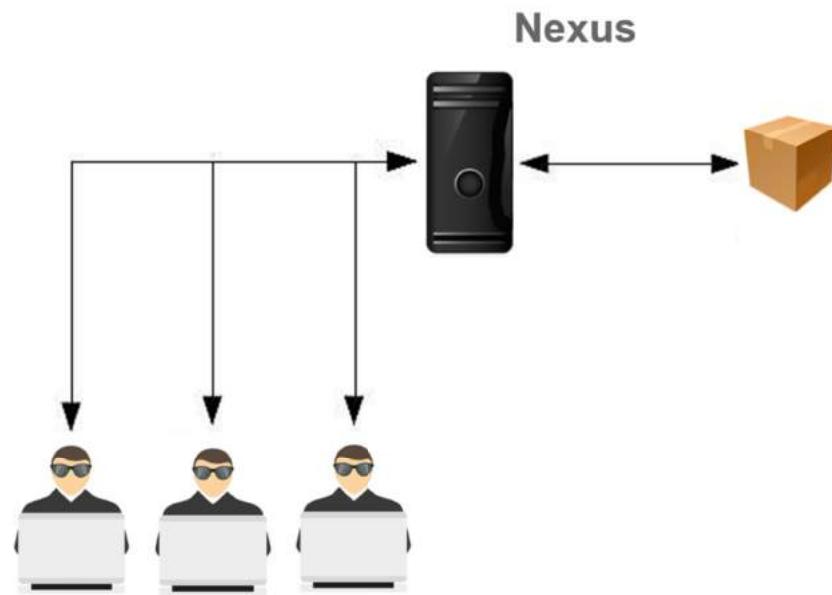
### Cache:

Agileza o processo de build das aplicações, permitindo uma independência De repositórios remotos.

### Adaptação:

Suporta diferentes formatos de pacotes como PiPy, Yum, Docker, Maven, Ant, Bower, Node/Npm, RubyGems e outros formatos desenvolvidos pela comunidade como R, CPAN e Apt.

## Porque usar



### Sonatype Nexus

1º – Construção Rápida. Armazenando os artefatos em um repositório local, permite o rápido acesso pelos desenvolvedores, agilizando o processo de aquisição dos mesmos que teriam de ser baixados para cada um que realizasse build do projeto.

2º – Redução do consumo de banda. Em empresas de grande porte, com múltiplos projetos, o download dos mesmos pacotes é rotineiro. O repositório Nexus permite o armazenamento local, reduzindo o consumo de banda.

3º – Redução do Consumo de banda no repositório central. Com o número de repositórios locais crescendo, o projeto do repositório central passa a ter um menor consumo de banda, o que auxilia no lado financeiro.

4º – Estabilidade e Prevenção. Garantir que mesmo sem internet, sua empresa não dependa do repositório central Maven e, continue os trabalhos normalmente.

5º – Controle e Autoria. Ter maior controle dos pacotes utilizados e poder auditá-los quando necessário.

6º – Manter artefatos terceiros. O repositório também pode ser utilizado para manter artefatos que não seriam encontrados em alternativas públicas como o Maven Central, PyPi, Yum.

7º – Habilidade de manter repositórios locais. Permitindo maior agilidade no desenvolvimento.

8º – Habilidade de manter repositórios remotos. Re却itórios no Nexus tamb閙 podem ser externos.

9º – F醱il adaptação e utiliza玢o. O re却itório Nexus é de f醱il implementa玢o e pode ser feito em alguns poucos minutos.

## Instalação

1

O Nexus exige como pré-requisito, a instalação do Java 8 da Oracle:

```
# sudo add-apt-repository ppa:webupd8team/java  
# sudo apt-get update  
# sudo apt-get install oracle-java8-installer
```

2

Verifique a versão do Java:

```
# java -version
```

```
java version "1.8.0_60"  
Java(TM) SE Runtime Environment (build 1.8.0_60-b27)  
Java HotSpot(TM) 64-Bit Server VM (build 25.60-b23, mixed mode)
```

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Instalação

Podemos também clonar um repositório que já existe na web:

1

Download:

```
# wget https://sonatype.com/oss-thank-you-tar.gz
```

2

Descompactando:

```
# tar -xf nexus-3.3.2-02-unix.tar.gz
```

3

Executar o script com opção start | stop | restart | run :

```
# ./nexus run
```

```
# ls -1
LICENSE.txt
NOTICE.txt
bin
deploy
etc
lib
public
system
```

LICENSE.txt and NOTICE.txt: contém detalhes legais sobre a licença e os avisos de direitos autorais.

bin — Contém o script de inicialização do nexus, bem como os arquivos de configuração de inicialização.

etc — Contém arquivos de configuração.

lib — Contém bibliotecas binárias relacionadas ao Apache Karaf.

public — Contém recursos públicos da aplicação.

system — Contém todos os componentes e plugins que constituem o aplicativo.

## Alterando a porta padrão

O Nexus por padrão funciona na porta 8081, porém é possível alterar este padrão, através do arquivo de configuração.

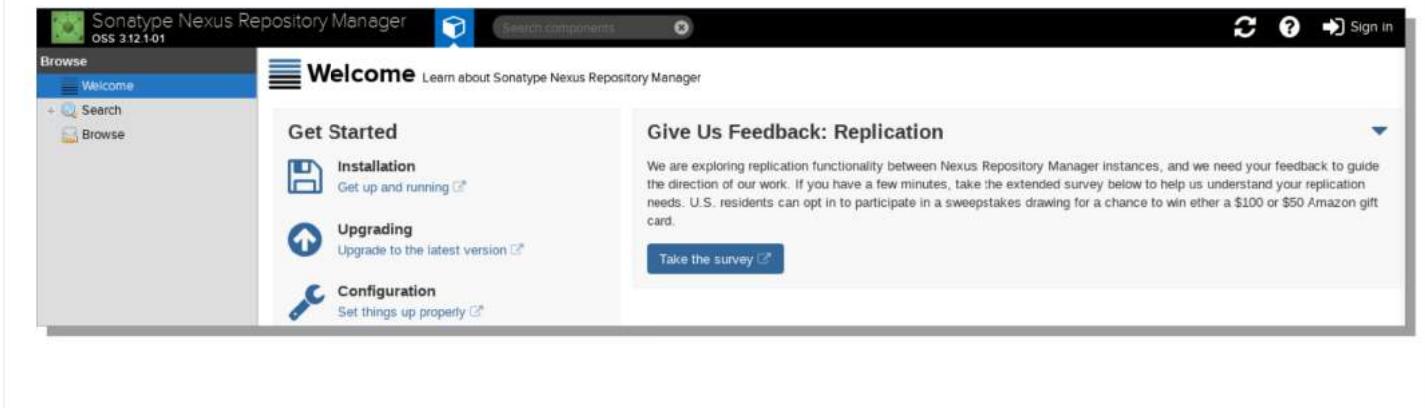
```
etc/nexus.properties
```

```
## Alterando a porta HTTP padrão para 9081 ##
application-port=9081
```

Dependendo da versão do SonaType Nexus, o nome do arquivo de configuração pode estar como nexus-default.properties.

## Página Inicial

`http://localhost:8081`



## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Formatos suportados



### Repository Formats

#### Pre-installed

Bower Docker Git LFS Maven .NET/NuGet Node/npm PyPI Raw RubyGems Yum

#### Community supported

Apt Conan CPAN ELPA Helm P2 R

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Login

A senha de admin padrão do Nexus é:

Login: **admin**

Senha: **admin123**

Clique em "Sign in" e entre com a senha de admin.



## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Criando Repositório Docker

Para criação de um repositório Docker onde armazenaremos as imagens geradas pela build final devemos adicionar um novo repositório.



Docker permite a configuração de três tipos de repositórios.

### **Hosted:**

Utilizada para armazenar um artefato criados pela equipe de desenvolvimento.

### **Proxy:**

*Utilizado para armazenar artefatos requisitados pela rede local para repositórios remotos.*

### **Group:**

Permite agrupar um conjunto de repositórios.

## Criando Repositório Docker

Para utilizar o Nexus como um repositório para imagens geradas no processo de build devemos adicionar um repositório do tipo *hosted*.

The screenshot shows the 'Repositories' section of the Sonatype Nexus web interface. At the top, there's a header with a database icon and the text 'Repositories' followed by a link 'Select Recipe'. Below the header is a table with a single column labeled 'Recipe ↑'. The table contains six rows, each with a small database icon and text: 'bower (group)', 'bower (hosted)', 'bower (proxy)', 'docker (group)', 'docker (hosted)', and 'docker (proxy)'. A blue arrow originates from the word 'hosted.' in the explanatory text above and points directly to the 'docker (hosted)' row in the list.

Recipe ↑
bower (group)
bower (hosted)
bower (proxy)
docker (group)
<b>docker (hosted)</b>
docker (proxy)

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Configurando Repositório Docker Hosted

The screenshot shows the 'Repositories' section of the Sonatype Nexus interface. A red box highlights the 'Name' field, which is empty and has a required status indicator. Below it, the 'Online' checkbox is checked. The 'Repository Connectors' section is expanded, showing fields for 'HTTP' and 'HTTPS' connectors, both currently empty. A note at the bottom of this section states: 'Connectors allow Docker clients to connect directly to hosted registries, but are not always required. Consult our documentation for which connector is appropriate for your use case.' The 'Force basic authentication' checkbox is also present.

Dentro da configuração do repositório podemos utilizaremos *connectors*, necessários para definirmos a porta de comunicação que o Docker deve utilizar quando realizando o *push* de uma imagem.

## Configurando cliente Docker para *Push* de Imagens

- 1 Configurando daemon.json:  

```
# vim /etc/docker/daemon.json
{
  "insecure-registries": ["192.168.56.3:8083"]
}

# systemctl restart docker
```
- 2 Autenticando cliente Docker:  

```
# docker login -u admin -p admin123 192.168.56.3:8083
Login Succeeded
```

Por padrão o cliente docker utiliza conexão com SSL (HTTPS), neste caso para que forcemos ele a confiar e utilizar o protocolo HTTP devemos reconfigurar o *dockerd* especificando dentro do arquivo *daemon.json* localizado em */etc/docker*.

Feita a configuração dentro do arquivo *daemon.json* podemos autenticar nosso cliente docker no repositório Nexus.

## Configurando cliente Docker para *Push* de Imagens

Agora com o docker autenticado podemos realizar o *push* da nossa imagem para o repositório do Nexus.

1 Criando Tag da image no Docker:

```
# docker tag aliflix-front 192.168.56.3:8083/aliflix-front
```

2 Realizando *push* da imagem para repositório:

```
# docker push 192.168.56.3:8083/aliflix-front
```

Agora com a imagem presente no repositório podemos fazer uso da mesma sempre referênciando o servidor Nexus, como por exemplo ao realizar um *pull* da image ou especificando no Dockerfile.

## Verificando imagem criada dentro do Nexus

The screenshot shows the Sonatype Nexus interface with the title "Browse" and the path "docker-hosted". Under "HTML View", the structure of the Docker repository is displayed:

- v2
  - aliflix-front
    - manifests
      - sha256:eb9919139d7ea6754260aea47772f8900775e8827e202a96f16e3352e9f388f8
      - sha256:ed766f811afc90e0ea7d7e990e77e7e4afab706f53f6e990e974d5eeeca2745bd
    - tags
      - 1
      - latest
  - blobs

Realizado o *upload* com sucesso, poderemos visualizar a imagem do Docker dentro do Nexus acessando o menu *Browse > Docker-Hosted*.

## Configurando Nexus para conexão HTTPS

1 Acessando diretório do certificado:

```
# cd /opt/sonatype-nexus/nexus-3.12.1-01/etc/ssl
```

2 Criando certificado:

```
# keytool -genkeypair -keystore keystore.jks -storepass password  
-keypass password -alias jetty -keyalg RSA -keysize 2048 -validity  
5000 -dname "CN=*.localhost, OU=Example, O=Sonatype, L=Unspecified,  
ST=Unspecified, C=US" -ext "SAN=DNS:nexus,IP:192.168.56.3" -ext  
"BC=ca:true"
```

3 Alterando formato do certificado:

```
# keytool -importkeystore -srckeystore keystore.jks -destkeystore  
keystore.jks -deststoretype pkcs12
```

### Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Configurando Nexus para conexão HTTPS

1

Definindo a porta SSL:

```
# cd /opt/sonatype-nexus/nexus-3.12.1-01/etc/
```

2

Editando arquivo nexus-default.properties:

```
# vim nexus-default.properties
...
application-port-ssl=8443
Nexus-args=...,${jetty.etc}/jetty-https.xml
```

3

Alterando senha do certificado:

```
# vim jetty/jetty-https.xml
<Set name="KeyStorePassword">password</Set>
<Set name="KeyManagerPassword">password</Set>
<Set name="TrustStorePassword">password</Set>
```

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

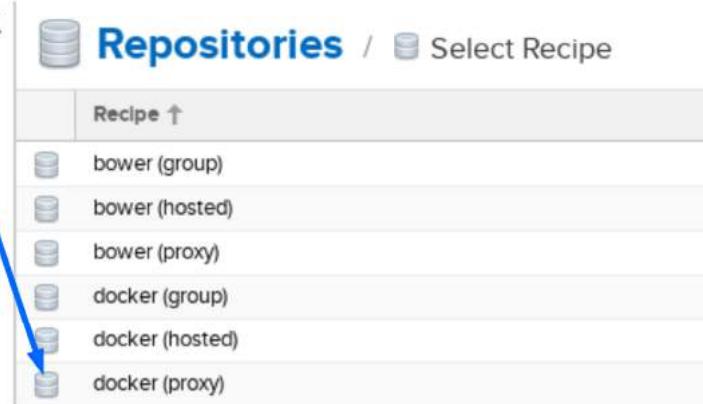
---

---

---

## Criando Repositório Docker Proxy

Para utilizar o Nexus como um repositório de imagens baixadas pelo Docker no processo de build devemos adicionar um repositório do tipo *proxy*.



The screenshot shows the 'Repositories' section of the Sonatype Nexus web interface. The title bar says 'Repositories' and 'Select Recipe'. Below is a table with the following data:

Recipe ↑
bower (group)
bower (hosted)
bower (proxy)
docker (group)
docker (hosted)
<b>docker (proxy)</b>

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

The screenshot shows the 'Repositories' section of the Sonatype Nexus interface. A new repository is being created with the name 'docker (proxy)'. The 'Name' field is required and has a red border. The 'Online' checkbox is checked. Under 'Repository Connectors', there are sections for 'HTTP' and 'HTTPS', each with a checkbox and a dropdown menu. The 'Force basic authentication' checkbox is checked. A note at the bottom states: 'Connectors allow Docker clients to connect directly to hosted registries, but are not always required. Consult our documentation for which connector is appropriate for your use case.'

Dentro da configuração do repositório podemos utilizaremos *connectors*, necessários para definirmos a porta de comunicação que o Docker deve utilizar quando realizando o *pull* de uma imagem.

Nesse caso iremos utilizar a porta 8084 no *connector HTTPS*.

## Configurando Repositório Docker Proxy

The screenshot shows the 'Repositories' section of the Sonatype Nexus web interface. Under 'Docker Registry API Support', there is a checkbox for 'Enable Docker V1 API' which is unchecked. Below this, under 'Proxy', there is a 'Remote storage' field containing 'Enter a URL' with a red error message 'This field is required'. Under 'Docker Index', there are three radio button options: 'Use proxy registry (specified above)' (unchecked), 'Use Docker Hub' (checked), and 'Custom index' (unchecked). The 'Location of Docker index' field contains 'https://index.docker.io/'.

Como um repositório *Proxy* devemos especificar o endereço que o Nexus irá realizar a consulta ao receber uma requisição do cliente docker para baixar um imagem do Docker Hub.

Como endereço de registro informamos:

<https://registry-1.docker.io>

Como endereço do Docker Index deixamos:

<https://index.docker.io>

## Configurando cliente Docker para conexão SSL

1 Configurando daemon.json:

```
# vim /etc/docker/daemon.json
{
  "insecure-registries": ["192.168.56.3:8083", "192.168.56.3:8084"]
}

# systemctl restart docker
```

2 Autenticando cliente Docker:

```
# docker login -u admin -p admin123 192.168.56.3:8084

Login Succeeded
```

Reconfigurando o arquivo *daemon.json* informamos o Docker para confiar na conexão pela porta 8084. Com isso, ao realizar o *pull* de uma imagem o sistema irá confiar no certificado auto assinado e solitar ao Nexus que realize o download da imagem.

## Banco de dados são particularmente desafiadores

- ✓ Enorme volume de informações.
- ✓ Ciclo de vida dos dados do aplicativo difere de outras partes do sistema.
- ✓ Dificuldade em automatizar o processo de migração do banco de dados.



Gerenciar e organizar dados envolve uma série de desafios para execução de testes e para os processos de implantação, por duas razões: primeiro, a quantidade de informação envolvida geralmente é enorme. Os bytes alocados para codificar o comportamento do aplicativo, seu código-fonte e informações de configuração, são em geral, amplamente superados pelo volume de dados que registram o seu estado.

Em segundo lugar, o fato do ciclo de vida dos dados do aplicativo ser diferente de outras partes do sistema. Os dados do aplicativo precisam ser preservados, na verdade, geralmente os dados estão mais atualizados que os usados para criá-los e acessá-los. Os dados precisam ser preservados e migrados durante novas implementações ou reversões (rollbacks) de um sistema.

### Banco de dados são particularmente desafiadores

- ✓ O gerenciamento dos dados de testes pode ser problemático.
- ✓ Uso de dados em produção para executar testes pode ser problemático.



### Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Princípios e práticas importantes para lidar com dados

- ✓ Versione seu banco de dados e use uma ferramenta para gerenciar migrações automaticamente.
- ✓ Use Schema para compatibilidade com versões anteriores e posteriores.
- ✓ Testes devem criar os dados que se baseiam como parte do processo de configuração.

Tal como acontece com outras alterações no sistema, quaisquer modificações nos bancos de dados usados como parte do processo de build, deploy, teste e lançamento, devem ser gerenciadas por mecanismos automatizados. Isso significa que a inicialização do banco de dados e todas as migrações, precisam ser capturados como Scripts e verificados no controle de versão.

Devido ao seu ciclo de vida, o gerenciamento de dados apresenta uma coleção de problemas diferentes daqueles discutidos no contexto do pipeline de implementação.

### Alguns dos princípios e práticas mais importantes

Versione seu banco de dados e use uma ferramenta como o *DbDeploy* para gerenciar migrações automaticamente.

Esforce-se em manter a compatibilidade, tanto com versões anteriores como posteriores em alterações de schema, tornando possível separar problemas de implementação e migração de dados de problemas de implementação de aplicativos.

Certifique-se de que os testes criem os dados em que se baseiam, como parte do processo de configuração, que os dados sejam particionados para garantir que não afetarão outros testes que possam estar em execução ao mesmo tempo.

## Princípios e práticas importantes para lidar com dados

- ✓ Limite as configurações compartilhadas entre testes, apenas aos dados necessários para o aplicativo iniciar.
- ✓ Tente usar a API pública do aplicativo para configurar o status correto de testes sempre que possível.

### Alguns dos princípios e práticas mais importantes (continuação)

Restrinja o compartilhamento de configuração entre testes, apenas para os dados necessários para iniciar o aplicativo, talvez para alguns dados de referência mais abrangentes.

Tente usar APIs públicas de aplicativos para configurar o estado correto de testes sempre que possível.

Na maioria dos casos, não use dumps do conjunto de dados de produção para fins de teste. Crie conjuntos de dados personalizados, selecionando cuidadosamente um subconjunto menor de dados de produção ou ainda de testes de aceitação ou de capacidade.

## Princípios e práticas importantes para lidar com dados

- ✓ Procure não usar dumps do conjunto de dados de produção para testes.
- ✓ Crie conjuntos de dados personalizados, de um subconjunto menor de produção para testes de aceitação e de capacidade.

### Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---



## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Objetivos da Aula

- ✓ O que é.
- ✓ Selenium WebDriver e IDE.
- ✓ Katalon(Selenium IDE para FF55+)
- ✓ Configuração do Selenium.

### Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## O que é

Selenium é conjunto de ferramentas open-source, para automação de testes multiplataforma em aplicações Web.

Por ser multiplataforma, o Selenium, é executado em diversos browsers e diferentes sistemas operacionais.

É capaz de criar códigos de automação, nas mais diversas linguagens de programação: C#, Java, Javascript, Objective-C, Perl, PHP, Python, Ruby. Para a criação dos códigos de nossos testes, utilizaremos o Selenium WebDriver.

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Selenium WebDriver



## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

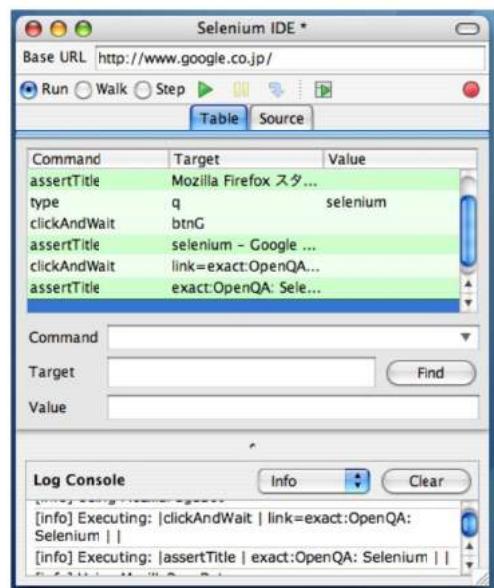
---

---

## Selenium IDE (Integrated Development Environment)

A maneira mais simples, para criarmos nossos testes automatizados, é utilizando o Selenium IDE.

Podemos facilmente gravar um cenário de teste, através de uma extensão do firefox, e exportar o código para a linguagem de script que desejarmos.



### Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Selenium IDE features

- ✓ Fácil reprodução e gravação.
- ✓ Seleção de campo inteligente, usará IDs, nomes ou Xpath, conforme necessário.
- ✓ Autocompletar, para os comandos mais utilizados do Selenium.
- ✓ Debug e breakpoints.

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

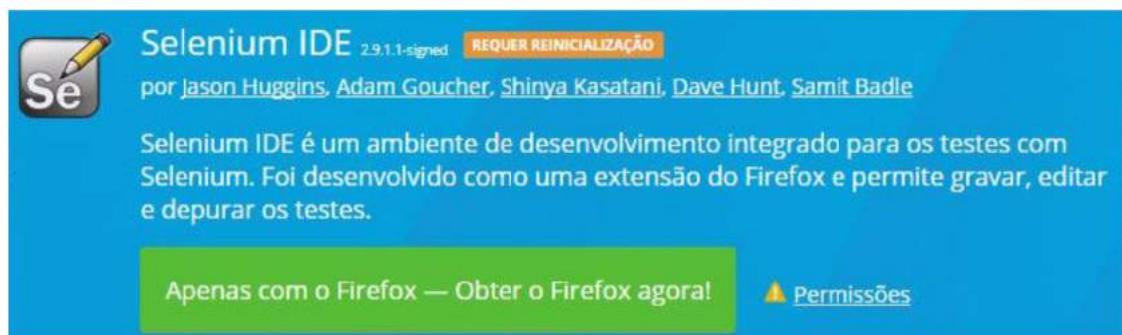
---

---

## Selenium IDE

Primeiramente, precisamos ter o **Firefox** instalado.

Baixe o **addon** para Firefox.



## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Selenium IDE

Nosso cenário de teste será o seguinte:

1. Entrar no site da 4Linux.
2. Pelo menu, clicar em "CURSOS".
3. Verificar se existe o título: "Cursos e Formações em Tecnologias Open Source".
4. Pelo menu, clicar em "CONTATO".
5. Verificar se existe o título "Entre em contato".

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Selenium IDE

Com o nosso cenário de teste, vamos automatizar a verificação de duas, das principais páginas da 4Linux, cursos e contatos, quanto a exibir corretamente o seu título.

Utilizaremos o Selenium IDE para gravar o nosso teste, após a gravação ser executada com sucesso, exportaremos o código que foi gerado para importar em nossa aplicação!

### Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Selenium IDE - Recording

Após a gravação do teste, exporte o código gerado, para rodar em nosso projeto Java. Maximize o Selenium IDE. Clique em File, em seguida, Export Test Case As, depois selecione a opção: Java / JUnit 4 / Web Driver

The screenshot displays two software interfaces side-by-side. On the left is the Selenium IDE, showing a recorded test case named 'Dexter...'. The test steps include opening the base URL 'https://www.4linux.com.br/' and performing various interactions like clicking links and asserting titles. On the right is the Katalon Automation Recorder, showing a recorded test suite named 'Untitled Test Suite'. This suite contains a single test case named '4linux' which mirrors the recorded steps from the Selenium IDE. Both interfaces have a toolbar at the top with various icons for recording, playing, and exporting.

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Projeto Aliflix

O projeto Aliflix é uma plataforma de filmes desenvolvida em linguagem Python 3 com framework Flask e banco de dados MongoDB, que é executado em ambiente Docker.



## Build do Jenkins para o Projeto Aliflix

### # Criando Imagem do Front e App

```
cd app  
docker build -t aliflix-python .  
cd ..  
cd frontend  
docker build -t aliflix-front .  
cd ..
```

### # Criando Imagem e executando o MongoDB

```
docker rm -f db || true  
docker run -tdi --name db -p27017:27017 mongo  
sleep 10
```

### # Alimentando banco

```
ip_db=$(docker inspect db | grep -w "IPAddress" | awk '{ print $2 }' | head -n 1 | cut -d "," -f1 | sed -e 's/\^"/g'  
export NODE_ENV=$ip_db  
sed -i -e 's/ip/"$ip_db"/g' MongoScripts/aula4.js  
mongo $ip_db:27017/dexter < MongoScripts/aula4.js  
sed -i -e 's/ip/"$ip_db"/g' MongoScripts/hello.js  
mongo $ip_db:27017/dexter < MongoScripts/hello.js
```

### # Rodando Containers

```
docker rm -f app || true  
docker run --name app --hostname app --link db -p5000:5000 -d aliflix-python  
docker rm -f front || true  
docker run -tdi --name=front --hostname front -p8085:80 aliflix-front
```

## Projeto Aliflix – Build Completa

Execute shell

```
Command docker login -u admin -p admin123 192.168.56.3:8084
cd app
docker build -t aliflix-python .
cd ..
cd frontend
docker build -t aliflix-front .
cd ..

docker rm -f db || true
docker run -tdi --name db -p27017:27017 192.168.56.3:8084/mongo
sleep 10
ip_db=$(docker inspect db | awk -F'"' '/"IPAddress":/ {print $4 ; exit}')
export NODE_ENV=$ip_db
sed -i -e 's/ip/"$ip_db"/g' MongoScripts/aula4.js
mongo $ip_db:27017/dexter < MongoScripts/aula4.js
sed -i -e 's/ip/"$ip_db"/g' MongoScripts/hello.js
mongo $ip_db:27017/dexter < MongoScripts/hello.js

docker rm -f app || true
docker run --name app --hostname app --link db -p5000:5000 -d aliflix-python
docker rm -f front || true
docker run -tdi --name=front --hostname front -p8085:80 aliflix-front
```

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Projeto Aliflix – Build em Etapas

The screenshot shows a Jenkins pipeline configuration. At the top, there's a toolbar with icons for Run, History, Configure, Add Step, Delete, and Manage. Below the toolbar, the pipeline is visualized as a sequence of colored boxes connected by arrows. The stages are:

- Pipeline #26 (Grey box)
- #26 aliflix-build (Green box, status: Success, duration: 5.2 sec)
- #22 aliflix-tests (Green box, status: Success, duration: 36 sec)
- #11 aliflix-qa-sonar (Green box, status: Success, duration: 39 sec)
- #7 aliflix-nexus (Red box, status: Failed, duration: 1.3 sec)
- aliflix-deploy (Blue box, status: N/A)

Arrows indicate the flow from Pipeline to #26, #26 to #22, #22 to #11, #11 to #7, and #7 to aliflix-deploy.

Em um Pipeline completo teremos as etapas de *Build*, *Test*, *Quality Analysis*, *Artifact Storage* e *Deploy*. Para uma melhor visualização utilizamos o plugin Build Pipeline que constrói uma visão de todos os estágios do nosso Pipeline CI/CD.

Para uma configuração desse tipo precisaremos configurar um FreeStyle Job para cada etapa do pipeline. Para isso devemos ter o repositório git configurado em cada uma e um processo de build adequado conforme segue.

### ===== BUILD STAGE =====

```
cd app  
docker build -t aliflix-api .  
cd ..
```

```
cd frontend  
docker build -t aliflix-front .  
cd ..
```

**===== TEST STAGE =====**

```
docker rm -f db || true
docker run -tdi --name db -p27017:27017 mongo
sleep 10

ip_db=$(docker inspect db | awk -F"" '/"IPAddress":/ {print $4; exit}' )
export NODE_ENV=$ip_db
sed -i -e 's/ip/"$ip_db"/g' MongoScripts/aula4.js
mongo $ip_db:27017/dexter < MongoScripts/aula4.js
sed -i -e 's/ip/"$ip_db"/g' MongoScripts/hello.js
mongo $ip_db:27017/dexter < MongoScripts/hello.js
```

```
docker rm -f app || true
```

```
docker run --rm --name app --hostname app --link db -p 5000:5000 -d aliflix-api
```

```
docker exec app nosetests --with-xunit /srv/app_test.py
docker cp app:/nosetests.xml ${WORKSPACE}
```

**===== QA Sonar =====**

Install 'SonarQube Scanner for Jenkins' and 'Sonar Quality Gates Plugin' plugins

**===== Nexus OSS =====**

```
docker login -u admin -p admin123 192.168.56.3:8084
```

```
docker tag aliflix-api 192.168.56.3:8084/aliflix-api
docker tag aliflix-front 192.168.56.3:8084/aliflix-front
```

```
docker push 192.168.56.3:8084/aliflix-api
docker push 192.168.56.3:8084/aliflix-front
```

**===== DEPLOY STAGE =====**

```
docker run -tdi --name db -p27017:27017 mongo
sleep 10
```

```
ip_db=$(docker inspect db | awk -F"" '/"IPAddress":/ {print $4; exit}' )
export NODE_ENV=$ip_db
sed -i -e 's/ip/"$ip_db"/g' MongoScripts/aula4.js
mongo $ip_db:27017/dexter < MongoScripts/aula4.js
sed -i -e 's/ip/"$ip_db"/g' MongoScripts/hello.js
mongo $ip_db:27017/dexter < MongoScripts/hello.js
```

```
docker rm -f app || true
```

```
docker run --name app --hostname app --link db -p5000:5000 -d aliflix-api
```

```
docker rm -f front || true
```

```
docker run -tdi --name=front --hostname front -p8085:80 aliflix-front
```

## Java

No processo de build, informamos todos os passos de compilação da nossa aplicação.

Para a aplicação em Java, passaremos os comandos do Maven, para compilação, execução dos testes, e comando para gerar relatório do nível de cobertura de testes na aplicação, deploy no Wildfly, deploy no Sonar, deploy no Nexus.

O processo de rodar as migrations do banco de dados, já está configurado automaticamente, para serem executadas juntamente com o deploy.

Para configurar o projeto Java cicd-dexter seguimos os mesmos passos apresentados na apostila para consigurar as integrações com Gitlab, Jenkins, Sonar Scanner e Sonar Quality Gates. Para a integração com Sonatype Nexus será necessário seguir os passos apresentados a seguir.

## Configurações no Maven

Primeiro, precisamos apontar no arquivo de configuração global do Maven, as configurações do nosso servidor do Nexus.

Para isso, vamos inserir, configurações no arquivo settings.xml:

1

Abrir o arquivo xml:  
# vim .m2/settings.xml

### Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

```
<?xml version="1.0" encoding="UTF-8"?>
<settings xmlns="http://maven.apache.org/SETTINGS/1.1.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/SETTINGS/
1.1.0 http://maven.apache.org/xsd/settings-1.1.0.xsd">

  <servers>
    <server>
      <id>nexus-snapshots</id>
      <username>admin</username>
      <password>admin123</password>
    </server>
    <server>
      <id>nexus-releases</id>
      <username>admin</username>
```

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

```
<password>admin123</password>
</server>
</servers>

<mirrors>
  <mirror>
    <id>central</id>
    <name>central</name>
    <url>http://localhost:8081/repository/maven-group/</url>
    <mirrorOf>*</mirrorOf>
  </mirror>
</mirrors>

</settings>
```

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Configurações no Maven

Após definir de forma geral, as configurações globais de acesso ao Nexus, precisamos definir em nível de projeto duas coisas:

1. Qual repositório do Nexus utilizaremos para baixar as dependências do projeto.
2. Qual repositório do Nexus usaremos para guardar o artefato gerado da nossa aplicação.

Ambas as configurações, serão realizadas no arquivo pom.xml, que fica na raiz da aplicação.

### Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Configurações no Maven

1 Adicione as seguintes configurações dentro APENAS da tag <project>:

# vim pom.xml

```
<repositories>
    <repository>
        <id>maven-group</id>
        <url>http://your-host:8081/repository/maven-public/</url>
    </repository>
</repositories>

<distributionManagement>
    <snapshotRepository>
        <id>nexus-snapshots</id>
        <url>http://your-host:8081/repository/maven-snapshots/</url>
    </snapshotRepository>
    <repository>
        <id>nexus-releases</id>
        <url>http://your-host:8081/repository/maven-releases/</url>
    </repository>
</distributionManagement>
```

### Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

```
<repositories>
  <repository>
    <id>maven-group</id>
    <url>http://your-host:8081/repository/maven-public/</url>
  </repository>
</repositories>

<distributionManagement>
  <snapshotRepository>
    <id>nexus-snapshots</id>
    <url>http://your-host:8081/repository/maven-snapshots/</url>
  </snapshotRepository>
  <repository>
    <id>nexus-releases</id>
    <url>http://your-host:8081/repository/maven-releases/</url>
  </repository>
</distributionManagement>
```

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Configurações no Maven

1 Na raiz do projeto, execute o seguinte comando maven:

```
# mvn clean deploy
```

A build deve ser executada com sucesso exibindo informações de Uploaded e Uploading. Caso execute pelo Jenkins acesse o console de saída da build para visualizar tais informações.

### Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

```
Uploaded: http://localhost:8081/repository/maven-snapshots/com/example/demo/
0.0.1-SNAPSHOT/demo-0.0.1-20170704.205625-1.jar (14085 KB at 34606.2 KB/sec)
Uploading: http://localhost:8081/repository/maven-snapshots/com/example/demo/
0.0.1-SNAPSHOT/demo-0.0.1-20170704.205625-1.pom
2/2 KB

...
Uploaded: http://localhost:8081/repository/maven-snapshots/com/example/demo/
maven-metadata.xml (275 B at 9.3 KB/sec)
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 5.487 s
[INFO] Finished at: 2017-07-04T17:56:26-03:00

[INFO] Final Memory: 36M/343M
[INFO] -----
Finished: SUCCESS
```

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Java

### Chamar alvos Maven de alto nível

#### Goals

```
clean org.jacoco:jacoco-maven-plugin:prepare-agent install wildfly:redeploy -Pprod verify sonar:sonar
```

**clean** — Limpa a pasta target gerada pelo Maven.

**org.jacoco:jacoco-maven-plugin:prepare-agent** — Gera relatório de cobertura de código.

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## Java

**clean** — Limpa a pasta target gerada pelo Maven.

**org.jacoco:jacoco-maven-plugin:prepare-agent** — Gera relatório de cobertura de código.

**install** — Configurado para deploy no Nexus.

**wildfly:redeploy** — Faz novo deploy no servidor Wildfly.

**-Pprod** — Perfil do Maven para setar as configurações do ambiente de produção.

**verify sonar:sonar** — Deploy no sonar para análise de código.

Se os passos de integração do Jenkins com Sonar forma seguidos o parâmetro **verify sonar:sonar** é desnecessário.

## PHP

No processo de build, informamos todos os passos de compilação da nossa aplicação.

Para a aplicação em PHP, utilizaremos a opção, invocar o shell script. Assim atualizaremos as dependências da aplicação configuradas no composer, e ainda: atualização da aplicação no servidor web apache, rodar as migrations do banco de dados, rodar os testes do codeception, deploy no Nexus e no Sonar.

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

**Executar shell**

```
Comando composer install -n  
rm -rf /var/www/php/*  
cp -r * /var/www/php/  
cd /var/www/php  
php vendor/bin/phinx migrate  
php codecept run
```

[Veja a lista de variáveis de ambiente disponíveis](#)

**Executar shell**

```
Comando sh nexus-deploy.sh
```

[Veja a lista de variáveis de ambiente disponíveis](#)

**Executar shell**

```
Comando /home/debian/sonar-scanner-3.0.3.778-linux/bin/sonar-scanner -Dsonar.projectKey=codeception -Dsonar.sources=module/Application
```

[Veja a lista de variáveis de ambiente disponíveis](#)

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

## .NET

No processo de build, informamos todos os passos de compilação da nossa aplicação.

Para a aplicação em .NET, usaremos a opção, invocar o shell script, para compilar a aplicação e rodar os testes do MSTEST.

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

**Build**

**Executar shell**

Comando `cd PrimeService.Tests  
dotnet restore  
dotnet test`

[Veja a lista de variáveis de ambiente disponíveis](#)

**Executar shell**

Comando `sh nexus-deploy.sh`

[Veja a lista de variáveis de ambiente disponíveis](#)

**Executar shell**

Comando `/home/debian/sonar-scanner-3.0.3.778-linux/bin/sonar-scanner -Dsonar.projectKey=dotnet -Dsonar.sources=.`

[Veja a lista de variáveis de ambiente disponíveis](#)

## Anotações

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---