DevOps
Ambiente de Desenvolvimento e Operação

DevOps

Mini-currículos

Renato Seixas Esteves

Possui graduação em Bacharelado em Ciência e Tecnologia - BC&T pela Universidade Federal do ABC(2013), especialização em MBA em Gestão de Projetos com Práticas PMI pela Faculdade de Informática e Administração Paulista(2017) e ensino-medio-segundo-graupelo Colegio Objetivo Baixada(2006). Atualmente é Engenheiro de Software da IBM BRASIL IND.MAQ. E SERVICOS LTDA e Professor da Faculdade Impacta de Tecnologia.

Vanderson Gomes Bossi

Mestrando em Ensino de Ciências pela Universidade Cruzeiro do Sul; Especialista em Engenharia de Software pela Faculdade Impacta de Tecnologia e Tecnólogo em Informática pelo Instituto de Ensino Superior de Santo André IESA. Com mais de 10 anos de experiência em Desenvolvimento de Sistemas e Arquitetura de Software utilizando tecnologias como Java, PHP, .NET (c#) e Banco de Dados SQL Server atuando em empresas como Ticket Serviços, ThyssenKrupp Metalúrgica e General Motors do Brasil como Arquiteto de software. Atualmente professor na Faculdade Impacta de Tecnologia.

DevOps

Introdução

O nome desta disciplina é Ambiente de Desenvolvimento e Operação, mas ao longo deste período utilizaremos a abreviação DevOps por entender que este termo (que se tornou uma buzzword na área de tecnologia da informação) expressa bem os principais objetivos da disciplina.

No período anterior do curso você desenvolveu sua lógica de programação, aprendeu a desenvolver programas utilizando uma linguagem de programação, adquiriu os fundamentos de bancos de dados e desenvolveu soluções IoT (Internet of Things). Com elas você desenvolveu competências e habilidades essenciais para um profissional que irá atuar na análise e desenvolvimento de sistemas.

No período atual você irá desenvolver competências e habilidades para lidar com desafios adicionais nesta área.

Alguns dos desafios atuais no desenvolvimento de sistemas estão relacionados ao aumento da complexidade do software. Esta complexidade pode, por exemplo, estar relacionada às regras de negócios, às formas de interação com o usuário ou à necessidade de integração com outros sistemas. Cada vez mais, é necessário melhorar a colaboração dentro da equipe para que o desenvolvimento seja bem-sucedido, seja através de práticas ou ferramentas.

Há também os desafios na operação dos sistemas, uma vez que muitas soluções precisam atender diversos usuários simultaneamente, ter uma alta disponibilidade, ser capaz de lidar com grande variação na demanda e lidar com grandes volumes de dados.

Além disso, a necessidade de disponibilizar novas versões, seja para corrigir erros ou para adicionar novas funções, traz o desafio de implantar o software de forma rápida, efetiva e com baixo impacto na operação da versão atual do sistema.

A disciplina de Ambiente de Desenvolvimento e Operação (DevOps) irá capacitá-lo a enfrentar estes desafios. Você desenvolverá a competência de configurar ambientes colaborativos, locais e/ou em nuvem, para apoiar as atividades de análise, desenvolvimento e execução de sistemas. Você aprenderá a configurar e empregar máquinas virtuais, ambientes de desenvolvimento e gerenciadores de banco de dados; utilizará ferramentas de controle de mudanças; criará projetos de integração contínua.

Pontos principais

A disciplina de Ambiente e Desenvolvimento de Operações (DevOps) desenvolverá competências e habilidades para que o aluno possa lidar com os desafios no desenvolvimento, na operação e na implantação de sistemas de forma colaborativa e ágil.

1. DevOps

Este capítulo explica qual é o significado do termo DevOps.

- ✓ DevOps: *Development + Operations*

1.1. Produção de software

Um software para ser produzido envolve várias atividades tais como:

- **Análise**: Trata do entendimento das necessidades e da especificação do que o produto deve fazer para atendê-las.
- **Definição da arquitetura**: definir a estrutura e os componentes do produto para atender as especificações.
- **Programação**: Implementar os componentes em uma ou várias linguagens de programação
- **Testes**: Verificar se a implementação atende o que foi especificado.
- **Distribuição, implantação e operação**: Um produto de prateleira, envolve a distribuição do que foi desenvolvido, mas se for um software desenvolvido sob encomenda, envolve a implantação (instalação) seguida pela operação do sistema em funcionamento.

A sequência e duração das atividades dependem do PROCESSO de desenvolvimento e entrega de software.

1.2. Desafios na entrega do software

Há muitos desafios na produção de software. Além do domínio das diversas disciplinas (análise, arquitetura, programação, testes, distribuição, implantação e operação), podemos, também, destacar como principais desafios para a entrega do software:

- Produzir em equipe.
- Produzir no prazo necessário.

- Produzir dentro do limite de custo.

1.3. DevOps

Com o objetivo de resolver ou pelo menos minimizar as dificuldades citadas anteriormente as equipes de desenvolvimento estão utilizando ou aprendendo o DevOps.
Então, vamos começar a entender primeiramente o que significa esse termo:
DevOps é um termo que vem da junção das abreviações das palavras ***Development*** e ***Operations*** fazendo referência a Desenvolvimento e Operações para criar equipes multidisciplinares que utilizam práticas e ferramentas compartilhadas e eficientes.

DevOps é a união de pessoas, processos e produtos para habilitar a entrega contínua do valor para os usuários finais. As práticas essenciais de DevOps incluem planejamento ágil, integração contínua, entrega contínua e monitoramento de aplicativos.

Podemos dizer, também, que estas duas palavras foram escolhidas porque estão relacionadas diretamente com os desafios na entrega do software:

- Geralmente as equipes de desenvolvimento e operações envolvem profissionais com competências e habilidades distintas.
- A transição de um software em desenvolvimento para a operação não é uma operação única. O software passa por diversas correções e melhorias durante o seu tempo de vida, o que torna cada vez mais importante a agilidade e eficiência nesta transição.

Pontos principais

Atente para os tópicos a seguir. Eles devem ser estudados com muita atenção, pois representam os pontos mais importantes do capítulo.

- Produzir um software envolve diversas disciplinas (análise, arquitetura, programação, testes, distribuição, implantação e operação).
- A produção de software tem de lidar com os desafios relacionados a trabalhar em equipe e dentro de limites de custo e prazo.

- DevOps expressa uma forma de pensar orientada a fazer uma entrega contínua do produto de software

2. SCM Básico

Criação e alteração de um repositório de controle de versão – SCM (Source Control Management)

- ✓ SCM (Source Control Management): Criação e alteração de um repositório de controle de versão

2.1. O que é o controle do código-fonte?

O controle de versões ou controle do código-fonte monitora e gerencia alterações efetuadas no código. Há os Sistemas de Source Control Management (SCM – Gerenciamento de controle do código-fonte) que disponibilizam um histórico de execuções de desenvolvimento de código com o intuito de ajudar na resolução de conflitos durante a integração de diferentes atualizações do Sistema que está sendo desenvolvido ou atualizado. Simplificando o processo de desenvolvimento e tornando-se uma fonte centralizada para todo o código do projeto de software.

Tornando-se possível um desenvolvedor colaborar com a programação do código num projeto em equipe mesmo isolando o seu trabalho até que esteja pronto e solucionar rapidamente problemas ao identificar quem fez alterações e quais foram elas. (AMAZON, 2018)

2.2. O que é o Git?

O Git é um sistema de controle de versão de código aberto (VCS) distribuído que permite armazenar código, rastrear histórico de revisão, mesclar alterações de código e reverter para versões de código anteriores quando necessário. A cópia criada no seu repositório é conhecida como ramificação e utilizando essa ramificação é possível trabalhar no código independentemente da versão estável da base de código. Após a finalização das alterações pode-se armazená-las como um conjunto de diferenças, conhecido como uma confirmação, também, pode-se extrair confirmações de outros colaboradores para o repositório, enviar

confirmações para outras pessoas e mesclar confirmações de volta para a versão principal do repositório. (ALJORD; CHACON, 2009)

Toda vez que é feito um commit, o Git compara as versões anteriores com uma operação visualizável chamada diff. Se houve uma mudança de commits anteriores, o Git armazena um novo snapshot no repositório.

2.2.1. Quais são os benefícios do Git?

O Git tem vários benefícios importantes:

- **Controle histórico de alterações**: Pode-se analisar por um gráfico de como os commits mudaram ao longo do tempo, ver quando e por quem as alterações foram feitas e reverter para um commit anterior, se necessário. Esse histórico facilita a identificação e correção de erros.
- **Trabalho em equipe**: Possibilita compartilhar o código com os colegas de equipe para revisão. Os recursos de ramificação e revisão permitem o desenvolvimento simultâneo, ou seja, vários desenvolvedores podem trabalhar no mesmo arquivo e resolver as diferenças mais tarde.
- **Melhore a velocidade e a produtividade da equipe**: O Git facilita o acompanhamento de alterações no seu código pela equipe.
- **Disponibilidade e Redundância**: Git é um VCS distribuído, o que significa que não existe um único local central onde tudo é armazenado. Em um sistema distribuído, há vários backups caso necessite, então é possível trabalhar off-line e confirmar as alterações quando estiverem prontos.
- **Popularidade do Git:** O Git é suportado por muitos ambientes de desenvolvimento integrados (IDEs) e muitas ferramentas de desenvolvedores populares, incluindo AWS CodeCommit , Jenkins e Travis. Existem muitos recursos gratuitos do Git disponíveis, como a página de código aberto do Git. (ALJORD; CHACON, 2009)

2.3. O que é um Branch?

O Git não armazena dados como uma série de mudanças ou deltas, mas sim como uma série de snapshots (registro instantâneo) e quando é feito um commit (atualização do repositório) no

Git o sistema guarda um objeto commit que contém um ponteiro para o snapshot do conteúdo que foi colocado na área de seleção.

Para exemplificar iremos supor um diretório contendo três arquivos, após serem colocar na área de seleção e fazer o commit. Sendo que, colocar na área de seleção cria o checksum de cada arquivo e armazenar esta versão do arquivo no repositório Git (o Git se refere a eles como blobs), e acrescenta este checksum à área de seleção (staging 12con):

$ git add README test.rb LICENSE
$ git commit –m 'commit inicial do meu projeto'

Quando é criado um commit executando git commit, o Git calcula o checksum de cada subdiretório (neste caso, apenas o diretório raiz do projeto) e armazena os objetos de árvore no repositório Git. O Git em seguida, cria um objeto commit que tem os metadados e um ponteiro para a árvore do projeto raiz, então ele pode recriar este snapshot quando necessário. Seu repositório Git agora contém cinco objetos: um blob para o conteúdo de cada um dos três arquivos, uma árvore que lista o conteúdo do diretório e especifica quais nomes de arquivos são armazenados em quais blobs, e um commit com o ponteiro para a raiz dessa árvore com todos os metadados do commit.
Caso haja modificação do código fonte e fizer um commit (atualizar o repositório) novamente o próximo commit armazenará um ponteiro para o commit imediatamente anterior.
Em resumo: Um branch no Git é simplesmente um leve ponteiro móvel para um desses commits. O nome do branch padrão no Git é 12cone12. Quando inicialmente fez commits há um branch principal (12cone12 branch) que aponta para o último commit que foi feito sendo que cada vez que é realizado um commit ele avança automaticamente. (ALJORD; CHACON, 2009)

Pontos principais

Atente para os tópicos a seguir. Eles devem ser estudados com muita atenção, pois representam os pontos mais importantes do capítulo.

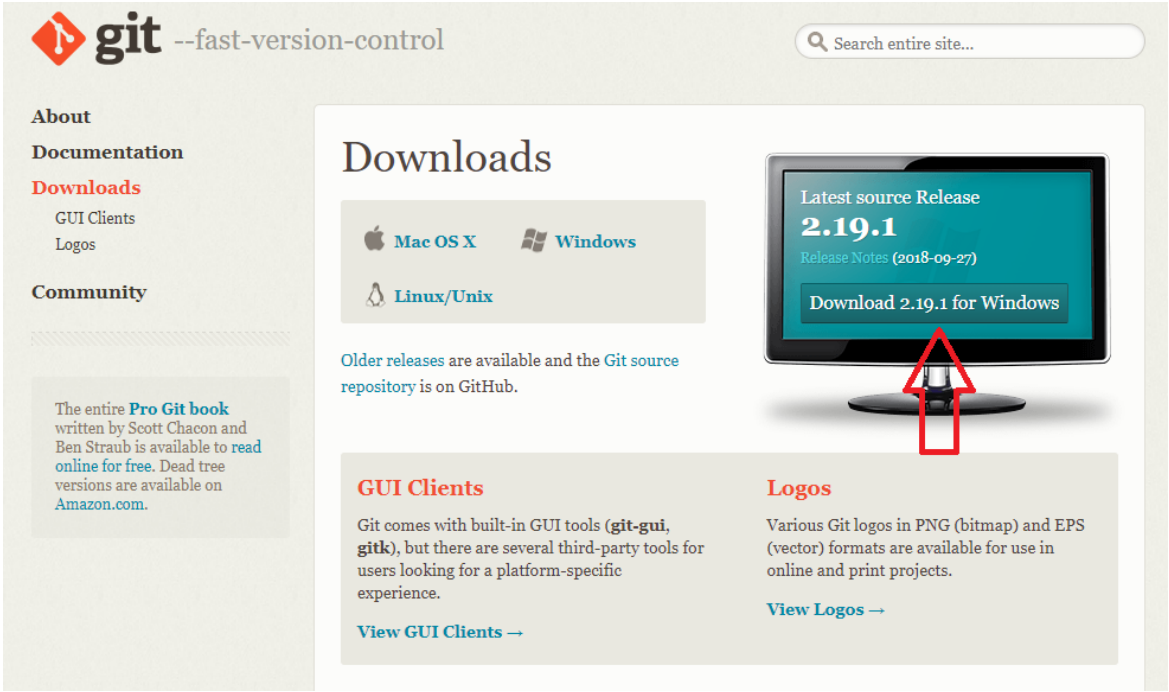
- Descobrir o que é o Git, que é um sistema de controle de versão de código aberto (VCS) distribuído que permite armazenar código, rastrear histórico de revisão, mesclar alterações de código e reverter para versões de código anteriores;
- Perceber que o Git tem vários benefícios importantes, como: Controle de histórico de alterações, trabalho em equipe, melhoria da velocidade e da produtividade da equipe, disponibilidade e Redundância e a popularidade do Git; e
- Entender como funciona um branch no Git que é um leve ponteiro móvel para um desses commits;

3. Instalando o GIT no Windows

Instalação passo a passo – SCM (Source Control Management) GIT

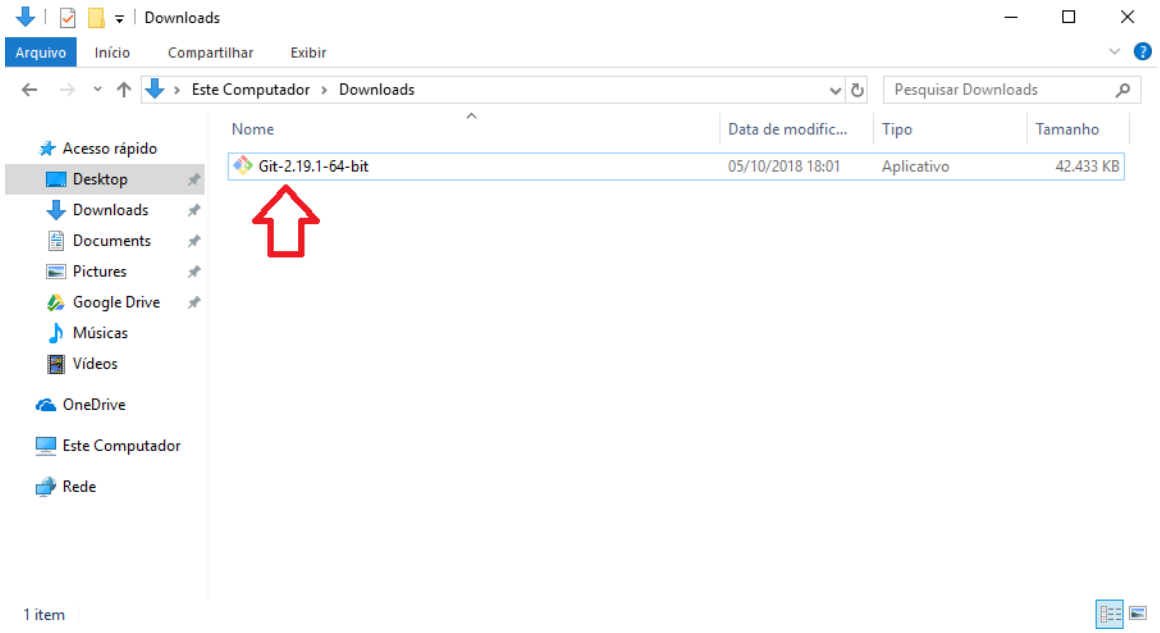
✓ GIT (Source Control Management): Instalação.

Primeiro, acesse o site oficial para baixar o instalador do git. Escolha o sistema operacional que você pretende instalar aqui iremos utilizar o Sistema Operacional Windows e efetue o download do instalador apropriado.



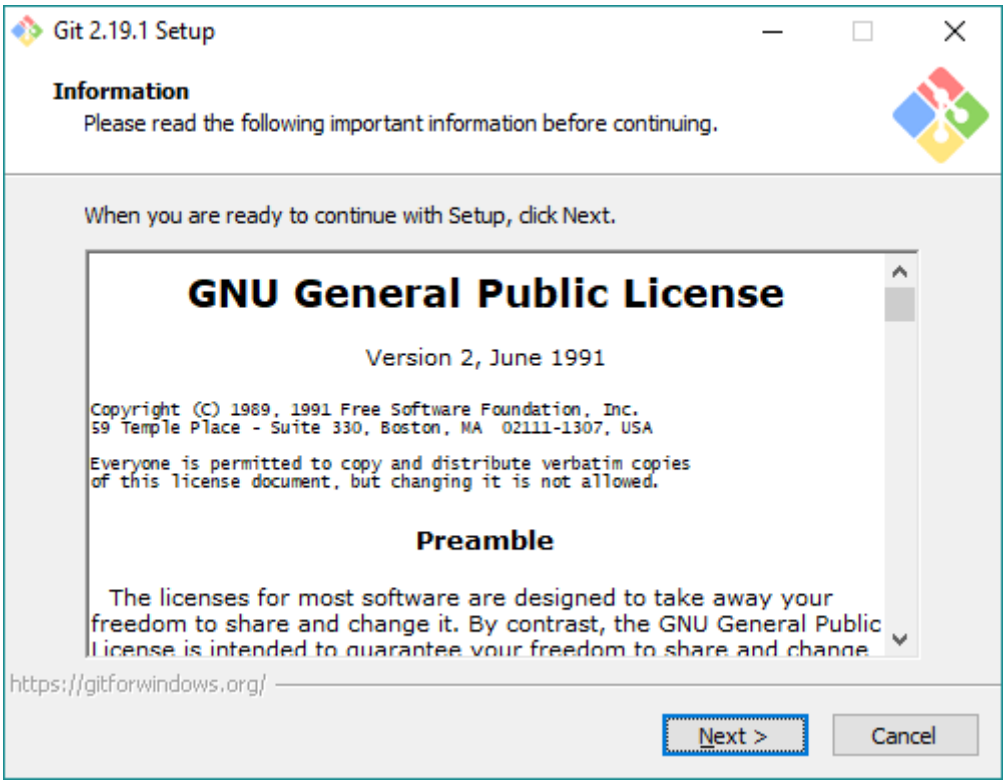
Fonte: Autor

Após o instalador ser baixado, execute-o.



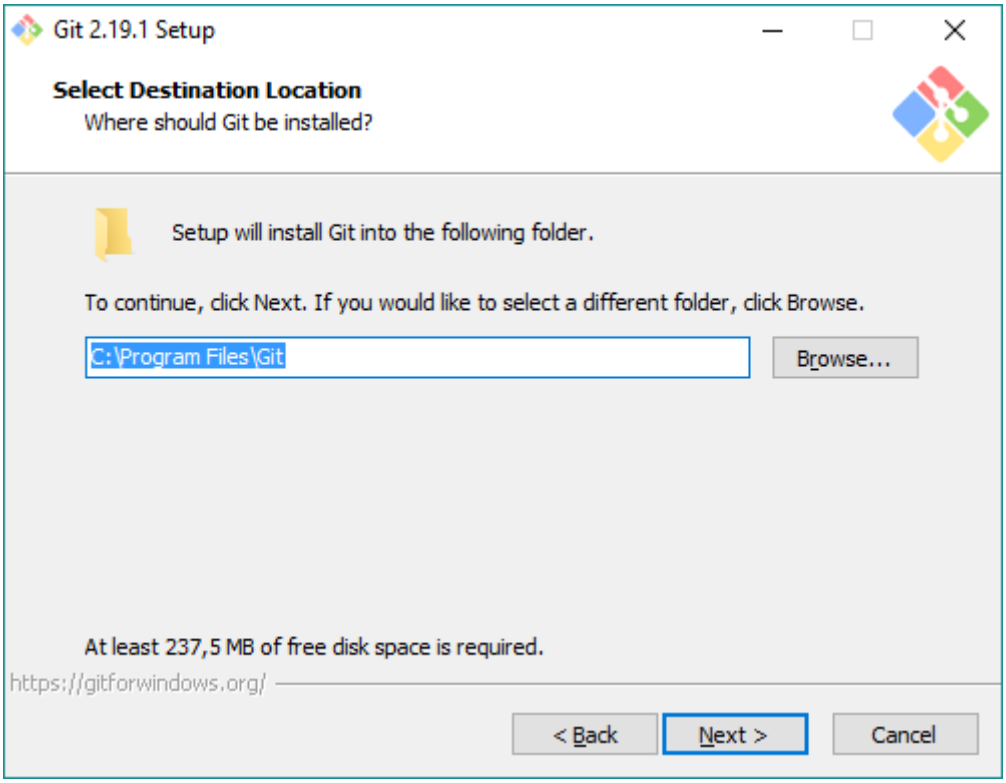
Fonte: Autor

A primeira tela do instalador do GIT apresenta os termos da licença de utilização do GIT. Basta clicar em Next.



Fonte: Autor

A segunda tela é para a seleção do local da instalação, eu mantenho o padrão C:\Program Files\Git, mas sinta-se livre para alterar. Após definir o local da instalação do GIT, clique em Next.

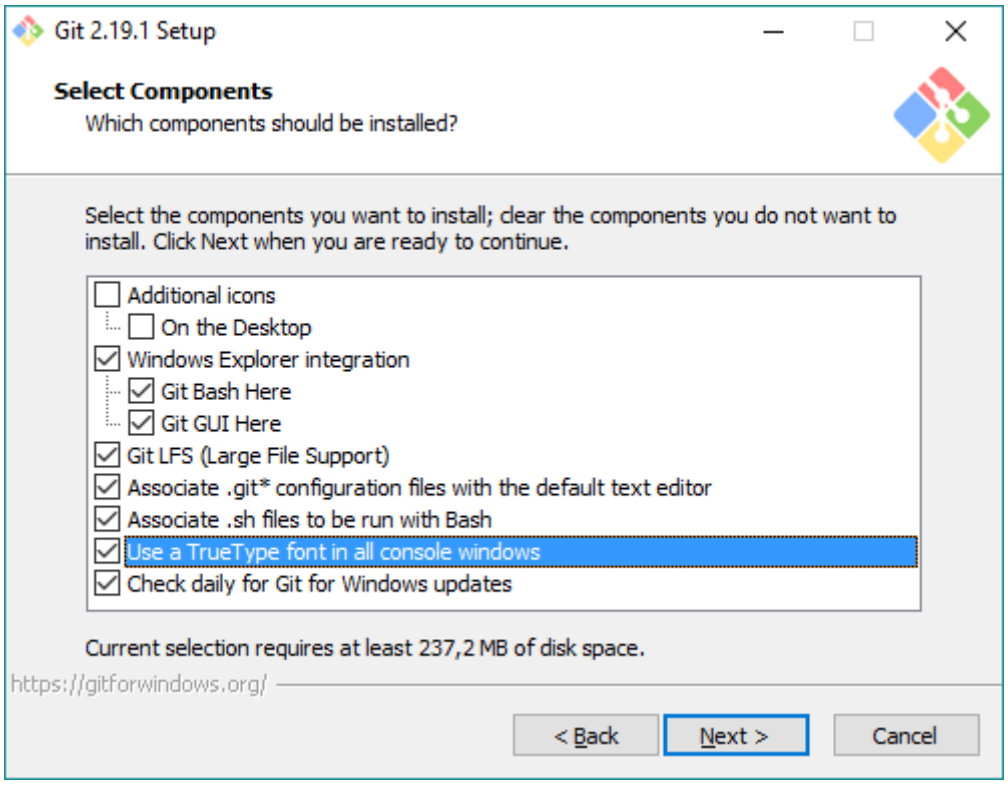


Fonte: Autor

Agora o instalador oferece a possibilidade de selecionar os componentes do GIT que queremos que seja instalado.

- ✓ Integração com Windows Explorer
- ✓ Suporte a arquivos grandes
- ✓ Associação de arquivos, e editor de texto padrão
- ✓ Permitir que o prompt de comandos seja colorido para as opções do git
- ✓ Verificação diária de atualização.

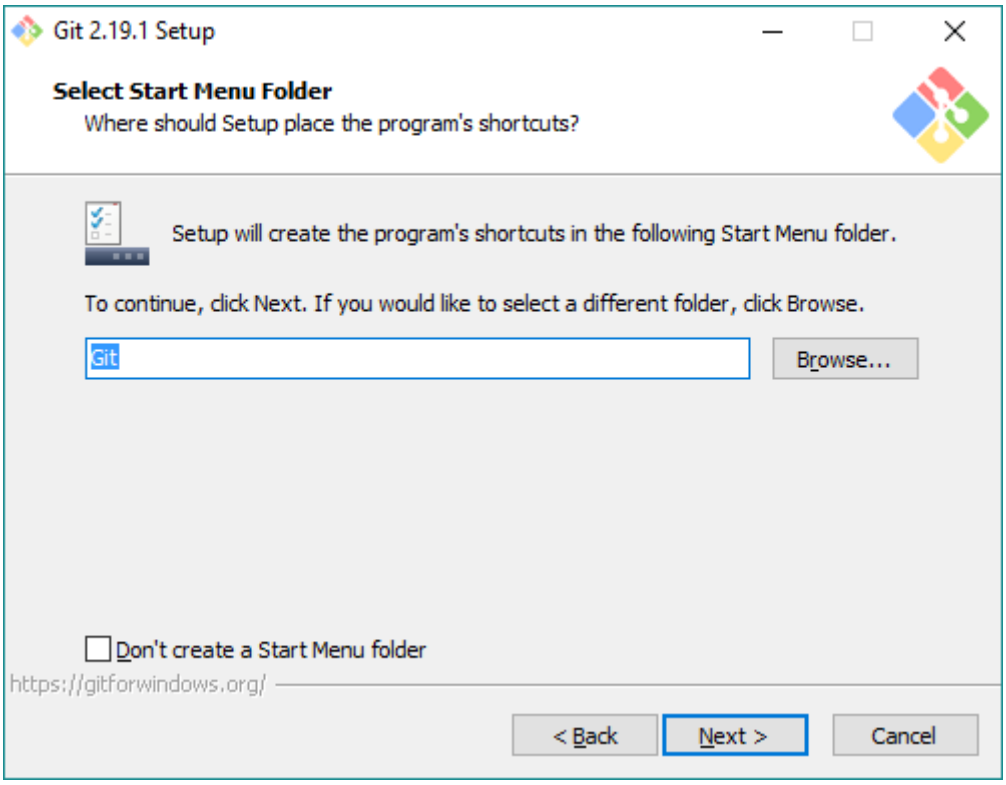
Escolha os componentes de sua preferência e clique em Next.



Fonte: Autor

Em seguida o instalador do GIT nos indica que criará uma pasta com atalhos no menu iniciar, se quiser, você pode marcar o checkbox Don't create a Start Menu folder para o instalador não criar esta pasta.

Clique em Next para continuar a instalação.

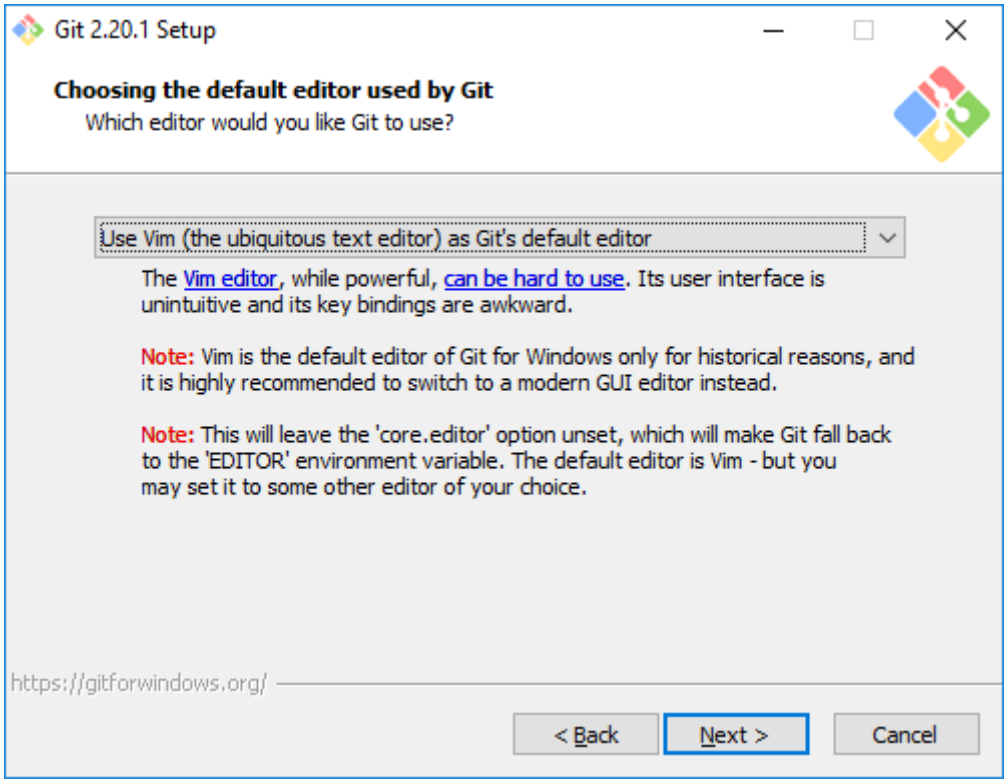


Fonte: Autor

Na sequência o instalador pede para selecionarmos o editor de texto que deve ser utilizado para editar os conflitos que por ventura acontecerem.

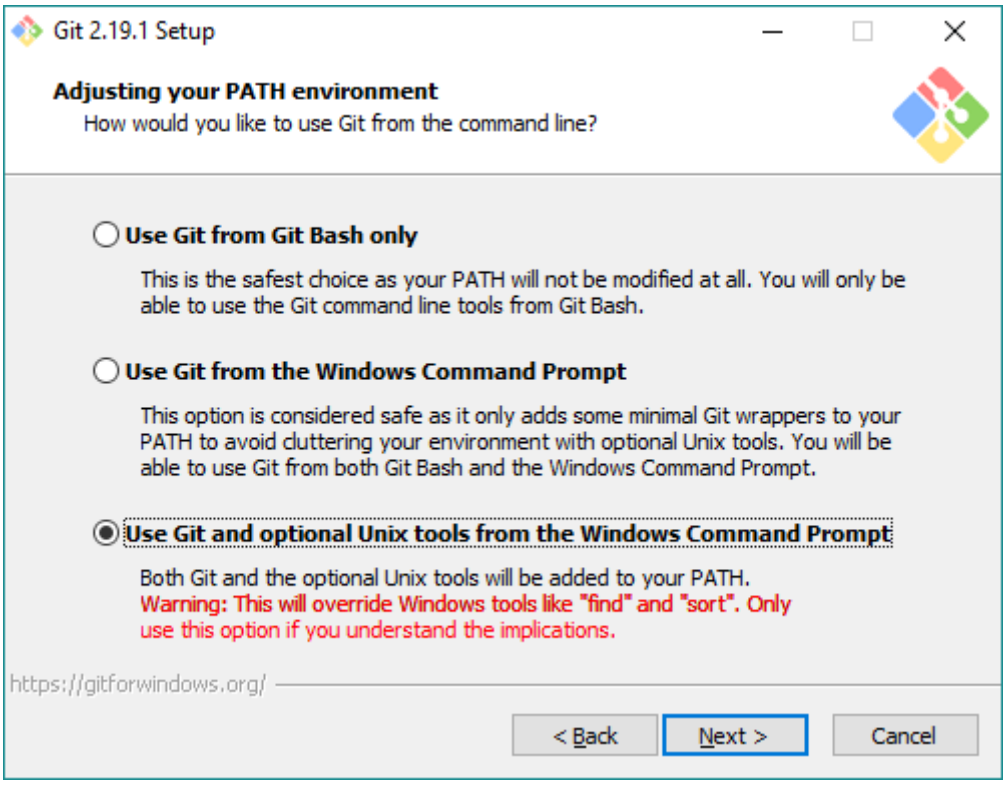
Escolha o editor que você tenha maior familiaridade, Notepad++, Sublime, Atom, VS Code, ou qualquer outro editor de sua preferência.

Após a escolha, clique em Next para continuar com a instalação.



Fonte: Autor

Na próxima tela o instalador do GIT nos pergunta se queremos usar os comandos do git:
Somente no prompt de comandos do próprio git (chamado de Git Bash), neste caso ele não vai alterar a variável de ambiente PATH.
No prompt do Windows (Windows Command Prompt), neste caso a variável de ambiente PATH será alterada para incluir o caminho de onde está o executável git.exe.
No prompt do Windows + comandos utilitários do línux. Esta opção é muito interessante, porque o instalador traz para o Windows alguns comando do Linux, como cat, ls, find, etc. Neste caso a variável de ambiente PATH será alterada para incluir o caminho do executável git.exe e dos executáveis de cada comando utilitário do linux.
Escolha a opção que acha mais adequada para você e clique em Next.

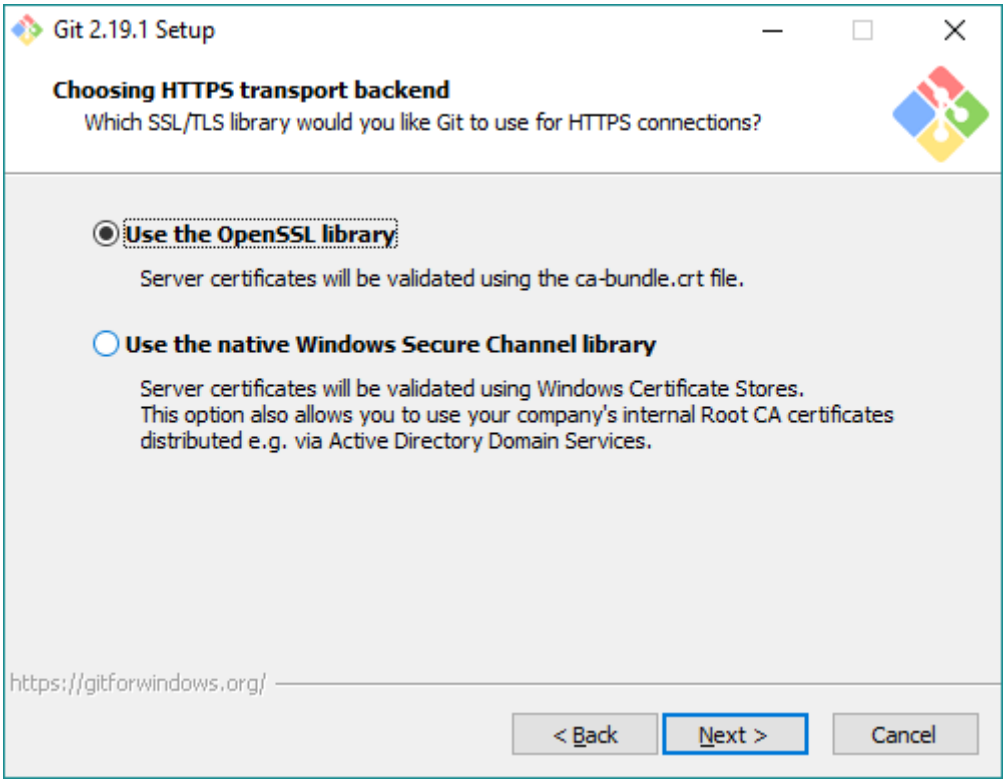


Fonte: Autor

A tela seguinte do instalador oferece a opção de escolher a biblioteca de validação de chaves de segurança SSL.

Geralmente utilizamos a OpenSSL, que é compatível com outras plataformas.

Escolha a que achar apropriada e clique em Next.



Fonte: Autor

Neste ponto da instalação, nos é perguntado como o git deve tratar o final dos a rquivos de texto.

3.1. Como é tratado o final de linha dos arquivos de textos

O Windows e o Unix, tratam o final de linha dos arquivos texto de formas diferentes.
O Windows segue um padrão antigo de comandos de impressora, chamado de CRLF, que indica para o cabeçote da impressora ir para o início horizontal (Carriage Return) e para iniciar uma próxima linha (Line Feed). Como se fosse aquela alavanca da máquina de escrever que funciona dessa forma, vai pro início e para a próxima linha.

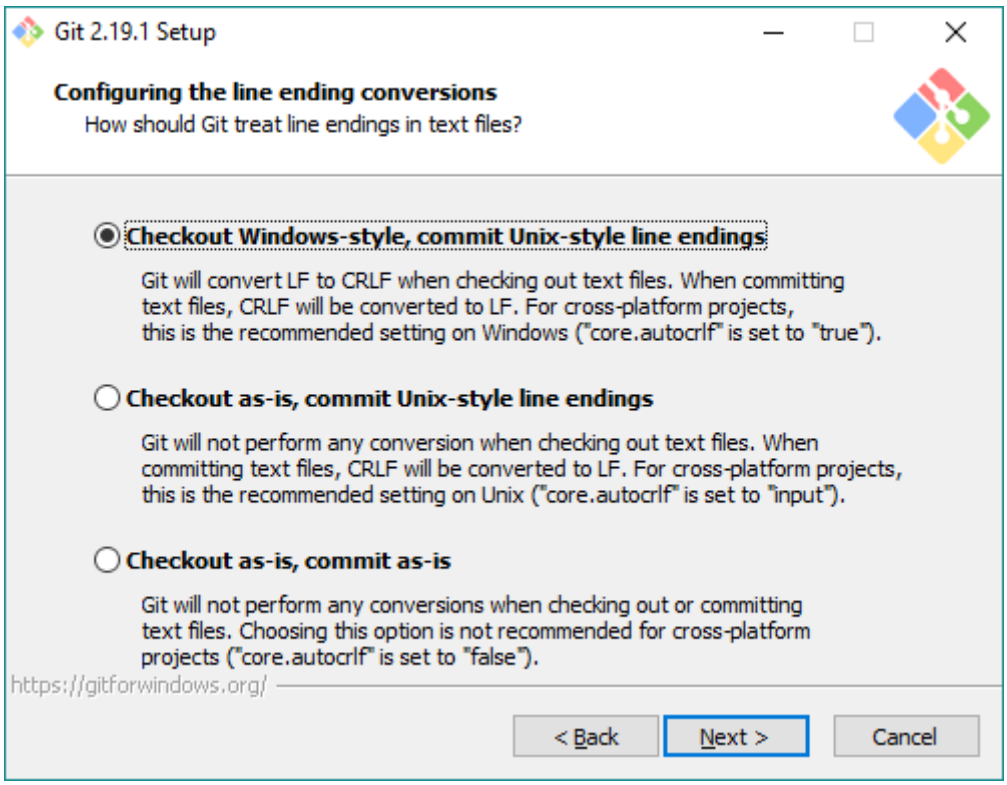
Já no linux, ficou comum somente o comando LF que teria o mesmo funcionamento do CRLF, ou seja, próxima linha...início.

Isso significa que o Linux usa um caracter ASCII para a quebra de linha e o Windows usa dois.

Entenda as opções que o instalador do GIT oferece:

- ✓ Converter LF para CRLF ao baixar arquivos e CRLF para LF ao comitar.
- ✓ Baixar como é (não converte nada), mas comitar convertendo CRLF para LF.
- ✓ Não converter nada, baixar como é e comitar como estiver.

Geralmente é marcado primeira opção, para evitar problemas de compatibilidade entre as plataformas.

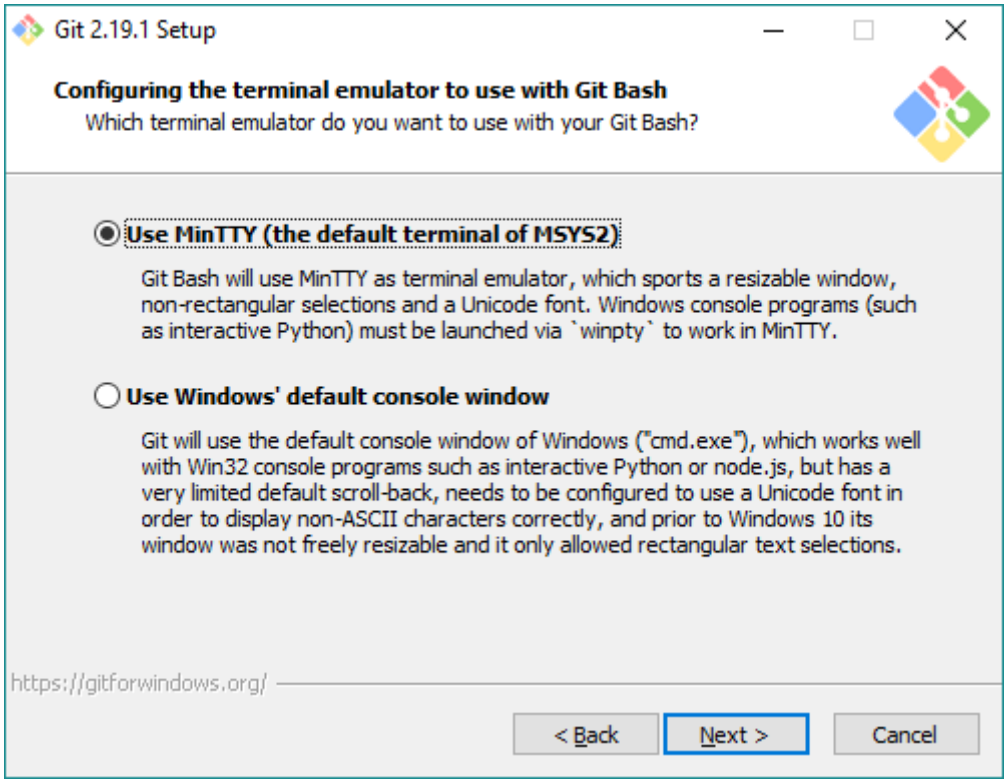


Fonte: Autor

Na sequência, o instalador oferece a opção de escolher o emulador de terminal (prompt) que queremos usar para o GIT.

Podemos usar o console padrão do Windows (cmd.exe) ou o MinTTY.

Em suma, o cmd é usado para compatibilidade com plataformas de 32 bits, ou seja, se o seu windows for 32 bits, é melhor escolher o cmd mesmo, mas se o seu windows for de 64 bits, é melhor escolher o MinTTY, porque ele oferece alguns recursos melhores para terminal, como por exemplo maximizar.

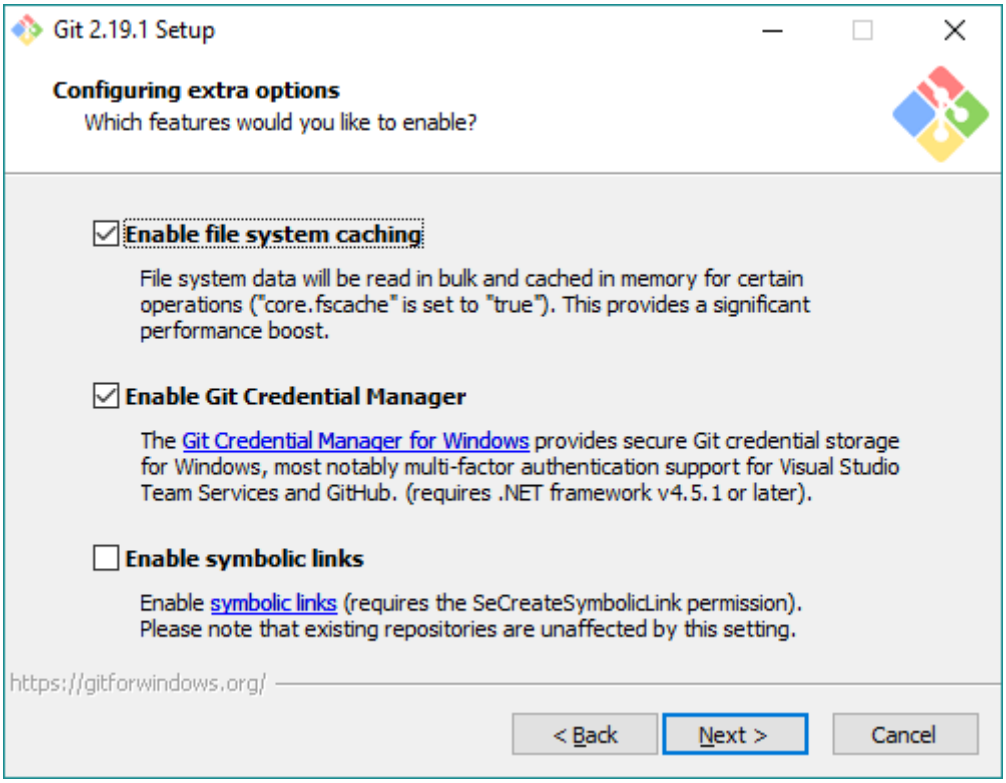


Fonte: Autor

A penúltima tela nos oferece algumas opções extras, são elas:

- ✓ Habilitar cache de arquivos na memória. Isso melhora o desempenho do git em alguns casos.
- ✓ Habilitar o gerenciador de credenciais do GIT. Isso permite autenticação em duas etapas no VSTS e no GitHub, e precisa do framework .NET 4.5 para funcionar.
- ✓ Habilitar links simbólicos.

Selecione os itens opcionais que você deseja e clique em Next.



Fonte: Autor

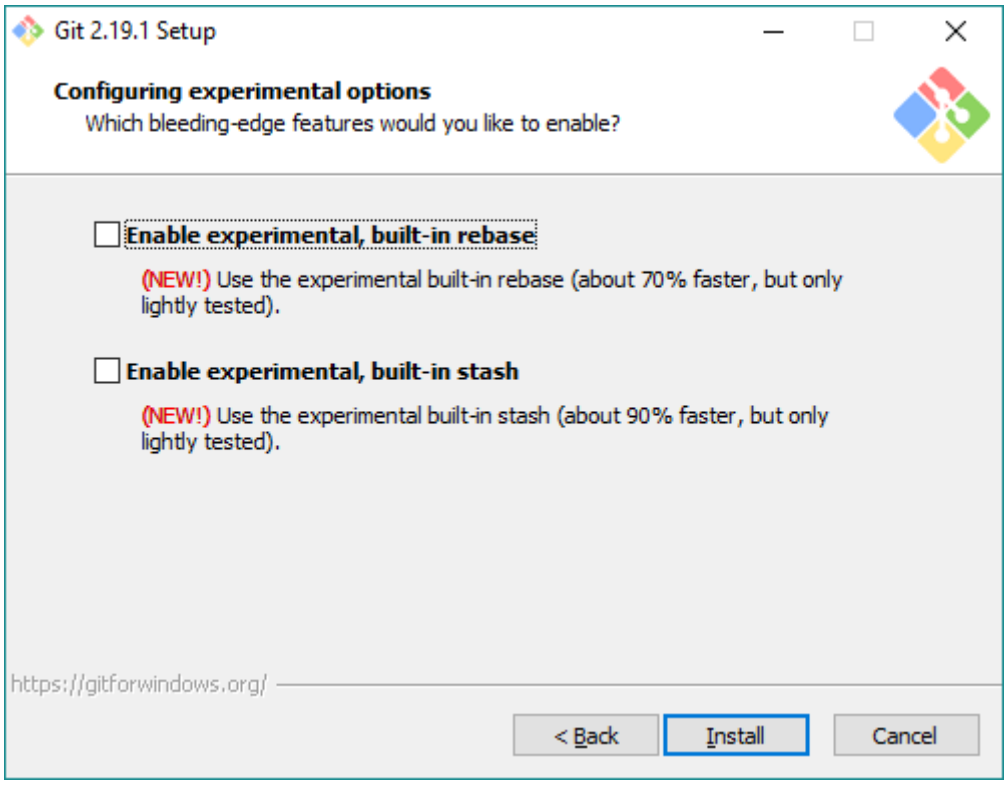
Por fim, última tela do instalador do GIT no Windows, o instalador nos oferece alguns componentes que estão em fase de experimentação.

São componentes que não foram muito testados, mas que parecem melhorar bastante a performance.

Você escolhe se quer instalar ou não ...

Como são componentes de em teste é, melhor não marcar.

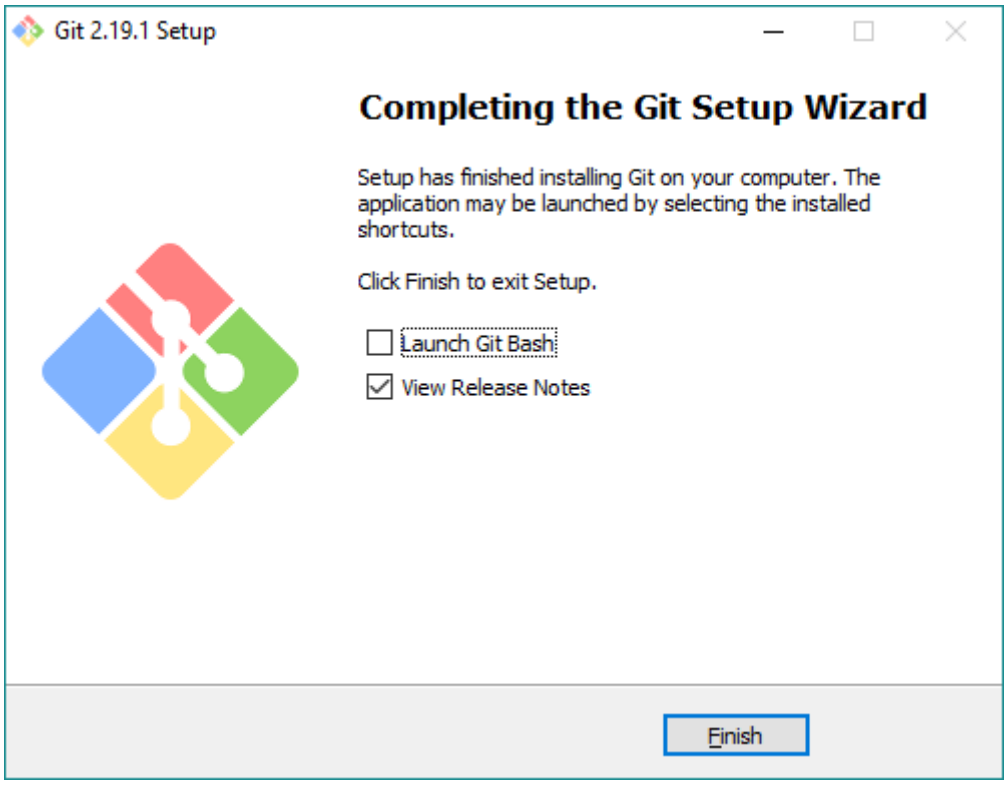
Por fim clique em Install para começar a instalação.



Fonte: Autor

Depois da instalação, o instalador oferece a opção de já rodar o prompt do GIT (Git Bash) e ver o arquivo de texto com as notas da versão lançada.

Para finalizar, clique em Finish.



Fonte: Autor

Pronto, agora você tem o GIT instalado no seu Windows!

Pontos principais

Atente para os tópicos a seguir. Eles devem ser estudados com muita atenção, pois representam os pontos mais importantes do capítulo.

- Como efetuar o download do software Git-ASC;
- Efetuar sua instalação no Sistema Operacional Windows e escolher as devidas opções;

4. SCM Intermediário

Criação e gerenciamento de branches – SCM (Source Control Management)

- ✓ SCM (Source Control Management): Criação e gerenciamento de branches

4.1. Gerenciamento de Branches

Veremos algumas ferramentas de gerenciamento de branches que são consideradas úteis.
O comando **git branch** faz mais do que criar e apagar branches ao executá-lo sem argumentos é possível listar os branches atuais, como por exemplo:

```
$ git branch
  iss53
* 27cone27
  testing
```

Note que o **caractere *** que vem antes do **branch 27cone27**: ele indica o branch que fez o checkout. Isso significa que se fizer um commit nesse momento, o branch 27cone27 irá se mover adiante com seu novo trabalho. Para ver o último commit em cada branch pode – se executar o comando git branch –v:

```
$ git branch –v
  iss53   93b412c consertar problema em javascript
* 27cone27  7ª98805 Merge branch 'iss53'
  testing 782fd34 adicionar scott para a lista de autores nos readmes
```

Outra opção útil para saber em que estado estão os branches é filtrar na lista somente branches que foi feito ou não o merge no branch que está trabalhando atualmente. As opções **–merged e –no-merged** estão disponíveis no Git desde a versão 1.5.6 para esse propósito. Para ver quais branches já foram mesclados no branch pode- se executar **git branch –merged**:

```
$ git branch –merged
  iss53
* 28cone28
```

Caso tenha feito o merge do branch iss53 antes, aparecerá na lista. Os branches nesta lista sem o * na frente em geral podem ser apagados com git branch –d; como já incorporou o trabalho que existia neles em outro branch não há problemas em 28cone-lo.
Para ver todos os branches que contém trabalho que ainda não fez o merge pode-se executar **git branch –no-merged**:

```
$ git branch –no-merged
  testing
```

Isso mostra outro branch. Por ele conter trabalho que ainda não foi feito o merge, tentar 28cone-lo com git branch –d irá falhar:

```
$ git branch –d testing
error: The branch 'testing' is 28cone28 ancestor of your current HEAD.
If you are sure you want to delete it, run `git branch –D testing`.
```

Se quiser realmente apagar o branch e perder o trabalho que existe nele pode forçar com –D, como a útil mensagem aponta. (ALJORD; CHACON, 2009)

4.2. Fluxos de Trabalho com Branches

Iremos abordar alguns fluxos de trabalhos comuns que esse tipo de criação de branches torna possível.

4.2.1. Branches de Longa Duração

Devido ao Git usar um merge de três vias, fazer o merge de um branch em outro várias vezes em um período longo é geralmente fácil de fazer. Isto significa que poderá ter vários branches que ficam sempre abertos e que são usados em diferentes estágios do seu ciclo de desenvolvimento.

Muitos desenvolvedores Git tem um fluxo de trabalho que adotam essa abordagem, como ter somente código completamente estável em seus branches 29cone29 — possivelmente somente código que já foi ou será liberado. Eles têm outro branch paralelo chamado develop ou algo parecido em que eles trabalham ou usam para testar estabilidade — ele não é necessariamente sempre estável, mas quando ele chega a tal estágio, pode ser feito o merge com o branch 29cone29. Ele é usado para puxar (pull) branches tópicos (topic, branches de curta duração) quando eles estão prontos, para ter certeza que eles passam em todos os testes e não acrescentam erros.

Os branches estáveis estão muito atrás na linha histórica de commits, e os branches de ponta (que estão sendo trabalhados) estão à frente no histórico.

Você pode continuar fazendo isso em vários níveis de estabilidade. Alguns projetos grandes podem ter um branch 'sugerido' (proposed) ou 'sugestões atualizadas' (pu, proposed updates) que contém outros branches integrados que podem não estar prontos para ir para o próximo (next) ou branch 29cone29. A ideia é que os branches estejam em vários níveis de estabilidade; quando eles atingem um nível mais estável, é feito o merge do branch acima deles.

Repetindo, ter muitos branches de longa duração não é necessário, mas geralmente é útil, especialmente quando se está lidando com projetos muito grandes ou complexos. (ALJORD; CHACON, 2009)

Pontos principais

Atente para os tópicos a seguir. Eles devem ser estudados com muita atenção, pois representam os pontos mais importantes do capítulo.

- Conhecer algumas ferramentas de gerenciamento de branches;
- Entender como funciona os Banches de Longa Duração;

5. SCM Avançado

Desenvolvimento colaborativo em um repositório – SCM (Source Control Management).

- ✓ Desenvolvimento colaborativo em um repositório;

5.1. GitHub – Contribuindo para um projeto

5.1.1. Projetos de bifurcação

Caso haja necessidade de alguém contribuir com um projeto existente para o qual não tem acesso **push**, poderá "bifurcar" o projeto isso quer dizer que o GitHub fará uma cópia do projeto. No GitHub, um "**fork**" é simplesmente o mesmo projeto em seu próprio namespace, permitindo que faça mudanças em um projeto publicamente como uma forma de contribuir de uma maneira mais aberta.

Dessa forma, os projetos não precisam se preocupar em adicionar usuários como colaboradores para fornecer acesso push. As pessoas podem bifurcar um projeto, empurrá-lo e contribuir com suas alterações de volta ao repositório original, criando o que é chamado de solicitação de extração. Isso abre um tópico de discussão com revisão de código, e o proprietário e o colaborador podem se comunicar sobre a alteração até que o proprietário esteja satisfeito com ela, quando o proprietário pode mesclá-la.

Para bifurcar um projeto, visite a página do projeto e clique no botão "Bifurcação", o 30cone é conforme a figura 01, no canto superior direito da página.

Após alguns segundos será direcionado para a nova página do projeto, com sua própria cópia gravável do código. (ALJORD; CHACON, 2009)

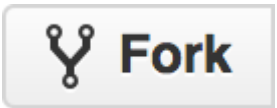


Figura 01 - O botão "Fork"

5.2. O fluxo do GitHub

O GitHub é projetado em torno de um fluxo de trabalho de colaboração específico, centrado em solicitações de pull. Esse fluxo funciona a equipe está unida em um único repositório compartilhado, ou uma empresa globalmente distribuída ou uma rede de estranhos contribuindo para um projeto através de dezenas de garfos. Ele está centrado no fluxo de trabalho de ramos do tópico, abordado no Git Branching .

Isso geralmente funciona:

- Bifurque o projeto
- Crie um ramo de tópicos de master.
- Faça alguns commits para melhorar o projeto.
- Empurre essa ramificação para o seu projeto do GitHub.
- Abra uma solicitação pull no GitHub.
- Discuta e, opcionalmente, continue comprometendo.
- O proprietário do projeto mescla ou fecha a solicitação pull.

Esse é basicamente o fluxo de trabalho do Integration Manager abordado no fluxo de trabalho do Integration-Manager, mas em vez de usar o email para comunicar e revisar as alterações, as equipes usam as ferramentas baseadas na Web do GitHub. (ALJORD; CHACON, 2009)

Pontos principais

Atente para os tópicos a seguir. Eles devem ser estudados com muita atenção, pois representam os pontos mais importantes do capítulo.

- Compreender a utilidade do botão Fork;
- Conhecer o fluxo do GitHub;

6. Qualidade

Automação de testes

- ✓ Definição e ferramentas de automação;

6.1. Definição

Automação de testes é utilizar um software para controlar a execução dos testes e a comparação dos resultados encontrados com os resultados esperados. É possível automatizar algumas tarefas de teste repetitivas que fazem parte do fluxo de teste de um software ou adicionar novos testes.

6.2. Quando não automatizar?

A automação de testes tem como objetivo reduzir esforços em tarefas repetitivas, antes de se inserir esse processo em um projeto deve-se analisar qual é o volume de testes que seu software necessita e o tempo que é gasto para executar essas baterias de testes deve-se verificar o esforço para inserir essa metodologia de testes que pode ser mais custoso do que a manutenção dos testes manuais, também deve-se verificar o tempo para a execução dos testes automáticos em comparação com os testes manuais.

Como qualquer outro software ele se comunica com o software que está sendo testado através de uma interface, então deve-se ter cuidado em modificar a interface pois os testes também sofrerão mudanças, onde muitas vezes inviabiliza a utilização dos testes automáticos.

6.3. Casos para automatização de testes

6.3.1. Testes de Carga e Desempenho

É impraticável que durante um teste necessite de mais de 100 (cem) pessoas para realiza-lo, então, os testes automáticos são muito bem-vindos neste caso.

6.3.2. Testes de Regressão

É uma das melhores aplicações da automação de testes. Testes de regressão usualmente custam muito tempo para serem executados manualmente e são muito suscetíveis a erro humano devido, ao seu nível de repetitividade, automatizando reduz-se drasticamente a possibilidade de o sistema ir para produção com a inserção de um novo defeito em uma funcionalidade antiga. Além disso, o ganho em tempo de execução dos testes geralmente justifica a sua utilização.

6.3.3. Testes manuais repetitivos

Testes manuais repetitivos normalmente exigem um grande nível de concentração de modo que ele não se distraia devido a um cansaço psicológico da repetição. Assim sendo, justifica-se a automação nos casos em que se verifica que seus testes têm obtido resultados não verdadeiros devido à falha humana.

6.3.4. Testes de Unidade

Um teste de unidade é aquele que testa uma única unidade do sistema. Ele a testa de maneira isolada, geralmente simulando as prováveis dependências que aquela unidade tem. Em sistemas orientados a objetos, é comum que a unidade seja uma classe.

6.4. Ferramentas de Automação

6.4.1. JUnit

O JUnit é um framework open-source, criado por Erich Gamma e Kent Beck, com suporte à criação de testes unitários automatizados na linguagem de programação Java.

O framework facilita na geração do código a realização dos testes automatizados com apresentação de resultados.

Os testes são executados rapidamente sem que, para isso, seja interrompido o processo de desenvolvimento. Ao final, o JUnit checa os resultados dos testes e fornece uma resposta imediata exibindo possíveis falhas.

Permite criar uma hierarquia de testes que possibilitará testar apenas uma parte do sistema ou todo ele.

O JUnit é amplamente utilizado pelos desenvolvedores da comunidade código-aberto.

6.4.2. Selenium

Selenium, ou Selenium WebDriver como também é conhecido, é um framework multi-plataforma para testes de interface em aplicações web desenvolvidos por Jason Huggins em 2004.

O software possui sua própria IDE (Selenium IDE) para gravação e execução de scripts de teste, que o usuário pode utilizar sem qualquer conhecimento de linguagens de programação. O Selenium IDE consiste em um plugin do navegador Firefox.

A gravação de testes funciona de maneira simples: basta executar o caso de teste manualmente enquanto o IDE armazena todos os passos seguidos.

O IDE permite o gerenciamento de vários casos de testes, além de viabilizar a criação de um plano de teste que possua vários casos de teste inclusos.

6.4.3. Jenkins

Jenkins é uma aplicação de código aberto escrita em Java que implementa o conceito de Integração Contínua.

O Jenkins atua monitorando a execução de tarefas do dia-a-dia de desenvolvimento alertando aos responsáveis caso algo inesperado ocorra.

Dessa forma, o Jenkins se foca basicamente em:

- **Construir builds de softwares continuamente** – onde se encarrega de gerar um build para seu software no intervalo de tempo desejado ou após algum evento desejado.
- **Rodar testes**
- **Executar tarefas externas** – possuindo completa integração com diversos sistemas operacionais, é possível programar o Jenkins para executar qualquer tarefa externa sendo ela local ou remota
- **Alertar usuários em casos de falhas** – ao executar qualquer das outras tarefas citadas acima o Jenkins pode ser utilizado para disparar um e-mail para os responsáveis pelos sistemas alertando possíveis falhas.

Para executar todas essas tarefas o Jenkins dispõe de uma base de milhares de plugins, os quais são utilizados para comunicar o servidor da aplicação com os outros serviços necessários.

Pontos principais

Atente para os tópicos a seguir. Eles devem ser estudados com muita atenção, pois representam os pontos mais importantes do capítulo.

- Compreender o uso das ferramentas de automação de testes.
- Conhecer as ferramentas de automação de testes.

7. Integração Contínua

Controle e versão e Integração Contínua.

- ✓ Controle de versão e Integração Contínua;
- ✓ Criação de projeto em um serviço de integração contínua e integração com o repositório de controle de código-fonte.

7.1. Para que serve a Integração Contínua?

Antigamente os desenvolvedores em uma equipe trabalhavam isoladamente e após um longo período juntavam suas alterações à matriz para concluir o projeto de software. Este processo de acumulação de alteração de código tornou-se muito difícil e demorado pois com isso também, acumulavam-se pequenos erros.

A proposta de integração contínua é uma prática de desenvolvimento de software de DevOps que traz um repositório compartilhado controlando a versão onde cada desenvolvedor executa teste em seus códigos antes da integração.

De acordo com Fowler (2006):

> A Integração Contínua é uma prática de desenvolvimento de software em que os membros de uma equipe integram seu trabalho com frequência, geralmente cada pessoa se integra pelo menos diariamente - levando a várias integrações por dia. Cada integração é verificada por uma compilação automatizada (incluindo teste) para detectar erros de integração o mais rápido possível. Muitas equipes acham que essa abordagem leva a problemas de integração significativamente reduzidos e permite que uma equipe desenvolva softwares coesos mais rapidamente.

7.2. Como Funciona a Integração Contínua

Será mostrado a seguir com alguns detalhes como a Integração Contínua funciona diariamente nos Projetos de Software:

Quando um mesmo projeto está sendo desenvolvido por vários programadores o desenvolvedor faz uma cópia de trabalho da última versão (atual) do projeto na sua máquina para completar a sua parte no projeto. Após o término é feito um build (desenvolvimento de uma versão), ou seja, o código fonte alterado é compilado e transformado em um executável e executa-se os testes automáticos.

Se não ocorrerem erros nos builds e nos testes coloca-se no repositório as alterações. Quando, por exemplo, dois desenvolvedores fazem parte do mesmo projeto e fizeram uma cópia do projeto em suas máquinas ao mesmo tempo mas um dos programadores terminou primeiro e atualizou o repositório (commit) e em seguida o segundo, também, terminou mas ao atualizar o repositório deverá levar em consideração as mudanças efetuadas pelo colega pois se as mudanças chocarem haverá erros tanto na compilação quanto nos testes, então para os erros não ocorrerem o segundo desenvolvedor deverá atualizar a cópia da sua máquina com as alterações efetuadas pelo colega e depois, fazer a build e a commit.

Após a commit é realizado um build novamente na máquina de integração baseada na versão principal do código e somente após esse build o trabalho terminou, porém não pode haver choque entre as versões. Sendo que, num build de integração não poderá haver falhas por muito tempo em um ambiente de Integração Contínua.

A integração contínua, também, cria e executa automaticamente testes de unidade nas novas alterações de código com o objetivo de detectar erros funcionais ou de integração imediatamente..

Os projetos envolvem muitos arquivos que necessitam serem organizados para se construir um software e manter esses arquivos atualizados quando envolve muitas pessoas é complicado, precisa-se de muito esforço, então as equipes de desenvolvimento criam ferramentas para gerenciá-los. Essas ferramentas são uma parte integrante de vários projetos de software (infelizmente ainda não são todos) e são conhecidas como ferramentas de gerenciamento de código fonte, gerenciamento de configuração, sistema de controle de versão, repositórios, entre

outros. O importante é usar um sistema de controle mesmo que a empresa tenha problema de orçamento pois há ferramentas open-source disponíveis.

Claro, o repositório somente irá funcionar se todos os envolvidos no projeto atualizarem adequadamente e todos os arquivos para fazer a build estiverem no repositório como: scripts de teste, arquivos de configuração, esquema de banco de dados, scripts de instalação, bibliotecas de terceiros e outros materiais importantes para os desenvolvedores, por exemplo: configurações de IDE. (FOWLER, 2006)

7.3. Quais são os principais objetivos da Integração Contínua?

Os principais objetivos da integração contínua são melhorar a qualidade, encontrar e investigar erros rapidamente, diminuir o tempo para lançar o software e suas atualizações. E, também, está relacionado a comunicação, onde todas pessoas envolvidas no projeto possam com facilidade ver o estado do sistema e as mudanças que tem sido realizada até o presente momento. Podemos, também, citar a redução de riscos que mesmo que a ferramenta não resolva os bugs gerados torna-se mais fácil encontrar e remover esses erros e isso se deve ao teste de unidade automático (já citado anteriormente). Sendo possível utilizar um diff debugging (avaliador de diferenças) onde compara as versões do sistema com uma versão anterior que não tinha o bug.

7.4. Algumas ferramentas de integração contínua

- **Jenkins**: CI é um mecanismo de integração contínua extensível.
- **Buildbot**: um sistema Python para automatizar o ciclo de compilação / teste para validar mudanças de código.
- **Tox**: uma ferramenta de automação que fornece empacotamento, teste e implementação de software Python diretamente do console ou do servidor de CI.

- **Travis**-CI: Um servidor de CI distribuído que cria testes para projetos de código aberto gratuitamente. Integra com perfeição ao GitHub e fornece funcionalidades para executar testes do Python.

7.4.1. Projeto utilizando a ferramenta Travis

Quando criamos um projeto em um serviço de Integração Contínua podemos utilizar a ferramenta Travis.

Para isso, adicionamos um arquivo, **travis.yml** ao seu repositório que geralmente é criado com este conteúdo de exemplo utilizando a linguagem Python:

```
linguagem: python
python:
  - "2,6"
  - "2,7"
  - "3,2"
  - "3,3"
# comando para instalar dependências
script: testes python / test_all_of_the_units.py
ramos:
  só:
    - master
```

Isso fará com que o projeto seja testado em todas as versões listadas do Python, executando o script fornecido e apenas criará o branch master. Há muito mais opções que pode ser ativado, como notificações, antes e depois de etapas e muito mais.

Pontos principais

Atente para os tópicos a seguir. Eles devem ser estudados com muita atenção, pois representam os pontos mais importantes do capítulo.

- Entender qual a proposta e a importância da Integração Contínua;

- Conhecer algumas ferramentas de Integração Contínua;
- Criar um projeto em um serviço de Integração Contínua.

8. Metodologias ágeis

Metodologias ágeis

- ✓ Infraestrutura ágil;
- ✓ Manifesto ágil;
- ✓ Ferramentas e equipe;
- ✓ Metodologias mais utilizadas.

8.1. Infraestrutura Ágil

A infraestrutura ágil é parte da cultura DevOps e têm como objetivo a redução do ciclo de vida do software propondo à equipe de programadores que desenvolvam pequenas partes do produto assim acelerando a entrega para o cliente com flexibilidade. (BOLINA et al. apud SEMEDO, 2017 <http://www.ietec.com.br/clipping/2018/01-janeiro/Agilidade-e-DevOps-uma-mistura-perfeita.pdf>)

Desde 2003 empresas começaram a conviver com ambientes virtualizados, então não havia mais espaço a infraestrutura da forma tradicional, sendo necessário pensar em infraestrutura como código.

8.2. Manifesto ágil

Segue os valores e princípios definidos pelo manifesto ágil, conforme publico na pagina http://www.manifestoagil.com.br/principios.html. Acesso em: 28 maio 2018:

- Nossa maior prioridade é satisfazer o cliente, através da entrega adiantada e contínua de software de valor.
- Aceitar mudanças de requisitos, mesmo no fim do desenvolvimento. Processos ágeis se adequam a mudanças, para que o cliente possa tirar vantagens competitivas.
- Entregar software funcionando com freqüencia, na escala de semanas até meses, com preferência aos períodos mais curtos.
- Pessoas relacionadas à negócios e desenvolvedores devem trabalhar em conjunto e diáriamente, durante todo o curso do projeto.

- Construir projetos ao redor de indivíduos motivados. Dando a eles o ambiente e suporte necessário, e confiar que farão seu trabalho.
- O Método mais eficiente e eficaz de transmitir informações para, e por dentro de um time de desenvolvimento, é através de uma conversa cara a cara.
- Software funcional é a medida primária de progresso.
- Processos ágeis promovem um ambiente sustentável. Os patrocinadores, desenvolvedores e usuários, devem ser capazes de manter indefinidamente, passos constantes.
- Contínua atenção à excelência técnica e bom design, aumenta a agilidade.
- Simplicidade: a arte de maximizar a quantidade de trabalho que não precisou ser feito.
- As melhores arquiteturas, requisitos e designs emergem de times auto-organizáveis.
- Em intervalos regulares, o time reflete em como ficar mais efetivo, então, se ajustam e otimizam seu comportamento de acordo.

8.3. Ferramentas e equipes

Há três tipos de ferramentas que combinadas originam a infraestrutura ágil, são elas:

- Orquestradores: Ferramentas para gerenciamento de configurações, permite executar comandos e controlar nodes/instancias, como Fabric, Capistano, Func e Mcollective.
- Ferramentas de gerência de configuração geralmente controlam estados de seu sistema, contribuem na centralização das configurações e facilitam a administração e criação de novos ambientes, como o Puppet, Chef, Cfegine e Salt.
- Ferramentas de bootstrapping auxiliar na instalação de um sistema operacional numa máquina física ou numa máquina virtual. Há alguns provedores de CLOUD como AWS e Rackspace e ferramentas como o Kickstart e Cobbler.

Apesar da qualidade em utilizar tais tecnologias as ferramentas sem a mudança de cultura das equipes de desenvolvimento muitos problemas ainda ficarão pendentes. É necessário que as práticas ágeis sejam implementadas em larga escala dentro da organização.

As equipes que trabalham com infraestrutura ágil também precisam de um método diferenciado de organização.

8.3.1. Metodologias mais utilizadas.

As metodologias ágeis mais comuns são: Extreme Programming (XP), Scrum, Lean Development, Feature-Driven Development (FDD), Kanban, RUP e OpenUP, mas o que o Scrum é o framework mais utilizado por ser o mais simples.

- Scrum: é um framework de desenvolvimento de software onde as pessoas podem resolver problemas complexos e adaptativos de forma empírica.
- Lean: O sistema de produção Lean surgiu da necessidade de melhoria do processo de produção das empresas automobilísticas japonesas, para manter a concorrência com as indústrias americanas.
- Método Kanban: É um método utilizado para a definição, gerenciamento e melhoramento dos serviços entregues. Sendo que, as tarefas são representadas através de cartões, que são fixados em um quadro

Pontos principais

Atente para os tópicos a seguir. Eles devem ser estudados com muita atenção, pois representam os pontos mais importantes do capítulo.

- A infraestrutura ágil é parte da cultura DevOps;
- Há três tipos de ferramentas que combinadas originam a infraestrutura ágil;
- As metodologias mais comuns são: Extreme Programming (XP), Scrum, Lean Development, Feature-Driven Development (FDD), Kanban, RUP e OpenUP, mas o que o Scrum é o framework mais utilizado por ser o mais simples.

9. Práticas ágeis

Práticas de metodologias ágeis.

- ✓ Práticas de metodologias ágeis;
- ✓ SAFe;
- ✓ Outras ferramentas.

9.1. Práticas de metodologias ágeis

Práticas de metodologias ágeis e as práticas de DevOps possuem um relacionamento profundo e acabam muitas vezes se misturando para suportar umas às outras essas duas coisas precisam caminhar juntas com o intuito de maximizar o resultado de produção das organizações.

Como já vimos nos capítulos anteriores, DevOps envolve todo o ciclo de vida do produto, mas a maioria das metodologias ágeis empregadas atualmente no Brasil focam-se apenas no desenvolvimento isso gera um problema, pois, é necessário de uma operação ágil capaz de acompanhar o desenvolvimento no todo.

Práticas ágeis se encontra atualmente num movimento mundial forte. As empresas trabalham com vários frameworks e metodologias ágeis.

9.2. SAFe

Agora, conheceremos um pouco sobre o SAFe.

SAFe é um conjunto de equipes ágeis que trabalham juntos de forma mais sincronizada e otimizada possível a fim de entregar software com qualidade, alinhados com os programas e estratégias da empresa. Os procedimentos de garantia de qualidade e de operações precisam trabalhar de forma contínua. E, tudo isso caracteriza o contexto rico para se aplicar as práticas de DevOps pois os objetivos das práticas de DevOps, como por exemplo, Continuous Delivery e Shift Left Testing, são bastantes similares com os objetivos ágeis colocados pelo framework.

O SAFE também é fortemente baseado no SCRUM e no pensamento Lean.

O monitoramento e a coleta de feedback contínuo é um elemento fortemente presente no SAFe em todo o ciclo de vida de execução das estratégias, dos programas e da produção contínua de softwares.

Embora a implementação, em sincronia, das práticas de DevOps e do SAFe tragam um desafio a mais para as organizações isso traz a maximização dos benefícios tais como a diminuição do custo de produção, o aumento da qualidade e a satisfação dos clientes.

Considerando, que a IBM atualmente oferta um dos conjuntos mais completos de solução para a implantação de DevOps atualizou essa solução incluindo um template de processo que suporta as práticas apresentadas pelo SAFe e outras ferramentas, como o IBM Urban Code, também podem ser utilizadas para quebrar silos se beneficiando de automação e incremento de visibilidade, colaboração e feedback entre times multifuncionais.

9.3. Outras ferramentas

- LESS (Large Scale SCRUM): O LESS foi compilado a partir das experiências de Craig Larman e Bas Voodle em consultoria e desenvolvimento de produtos em larga escala, tendo como pilares o SCRUM, o pensamento Lean e Teoria de Sistemas.
- Nexus: Teve origem a partir da experiência de Ken Schwabber, um dos pais do SCRUM, no desenvolvimento de produtos com vários times Scrum operando em paralelo. O Nexus é mantido pela comunidade Scrum.Org. É a sistematização da técnica de escala o Scrum chamada Scrum of Scrums, para projetos que demandem entre 3 e 9 times operando em paralelo. Ele introduz novos papéis e um time especial (Time de Integração) e tem como foco central a integração e coordenação dos esforços de cada time na construção dos incrementos de produtos.
- Holocracia: É um sistema descentralizado de tomada de decisões para que empresas possam criar mecanismos de auto-organização para operar em ambientes dinâmicos e complexos. A Holocracia tem em seus pilares conceitos sociológicos de organizações de centro vazio, conceitos de auto-organização, teoria da complexidade e teoria da emergência (emergence). A Holocracia não é um framework de TI e tampouco centrada em projetos apenas. Ele lida com o problema de criar organizações anti-frágeis (como definido por Nassib Taleb no seu potente livro Anti-Fragilidade: coisas que se beneficiam da desordem). A

Zappos, bilionária empresa americana recentemente comprada pela Amazon, implementa a holocracia como o seu sistema gerencial descentralizado. O portal Holacracy.org é uma vasta fonte de recursos para interessados e praticantes.

Cada uma desses frameworks possui dezenas de casos documentados e milhares de empresas em todo o mundo que os usam atingir objetivos econômicos e sociais.

Pontos principais

Atente para os tópicos a seguir. Eles devem ser estudados com muita atenção, pois representam os pontos mais importantes do capítulo.

- Conhecer algumas ferramentas como: LESEE, Nexus,SAFe e entre outros.

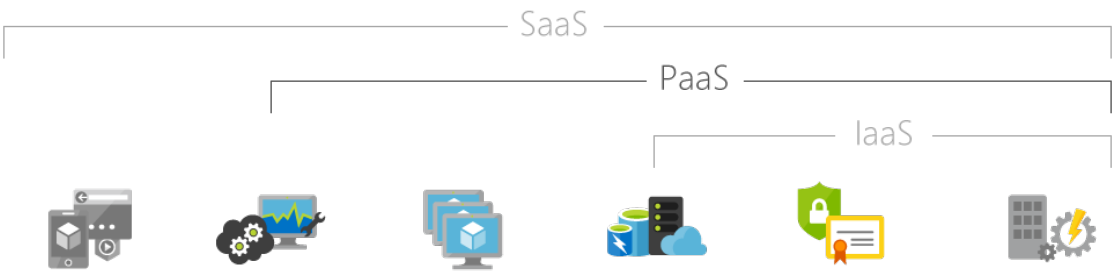
10. PaaS

Serviço de Plataforma de Nuvem - Paas

- ✓ Computação em Núvem;
- ✓ PaaS e seus benefícios.

10.1. Computação em Núvem

O **DevOps** se utiliza do **cloud computing (computação em núvem)**. E, com a Cloud há muitos serviços que facilitam essa metodologia, entre os mais comuns estão o SaaS, IaaS e PaaS.



10.1. Serviços Fonte: https://azure.microsoft.com/pt-br/overview/what-is-paas/

Para explicar melhor PaaS, será definido antes os outros dois conforme a divisão apresentada na figura 9.1.

- IaaS: Visualizando a figura da direita para a esquerda: Planta física Data Center, Segurança de rede e Servidores e armazenamento. Onde é possível utilizar servidores virtuais e outros dispositivos de infraestrutura ao invés de comprar servidores, roteadores, racks.
- SaaS: Inclui todos os itens do IaaS e também Sistemas Operacionais, Ferramentas de Desenvolvimento e Aplicativos hospedados e para utilizar algum software é pago pela utilização e não por sua licença.

10.1.2. PaaS

Serviços de plataforma de Nuvem (PaaS) são utilizados para aplicações que fornece componentes de nuvem ao software e fornecer infraestrutura como serviço, PaaS oferece as mesmas vantagens que o IaaS e seus recursos adicionais – middleware, ferramentas de desenvolvimento e outras ferramentas de negócios – dão ainda mais vantagens ao cliente.

Resumindo o PaaS, é um ambiente de desenvolvimento e implantação de aplicativos compartilhado. Esse ambiente oferece serviços que permitem criar bancos de dados, middlewares, serviços para realizar deploy, testes e manter as aplicações tudo isso no mesmo ambiente de desenvolvimento e é pago conforme o uso da plataforma

Há vários tipos de PaaS, incluindo público, privado e híbrido, sendo que inicialmente o PaaS foi concebido como público.

- PaaS público é derivado de software como serviço (SaaS) e gerenciamento do servidor é feito pelo provedor.
- O PaaS privado pode ser baixado e instalado tanto em infra-estrutura no local de uma empresa, ou em uma nuvem pública é capaz de organizar os componentes de aplicação e banco de dados em uma única plataforma de hospedagem.

Um exemplo de um serviço PaaS que podemos citar é o um servidor que hospeda site.

Sendo que o Windows Azure, além de IaaS e SaaS, também se encaixa nesta categoria por oferecer ao usuário a plataforma necessária para desenvolver as suas próprias aplicações sem que ele precise se preocupar com a infraestrutura por traz disto <https://azure.microsoft.com/pt-br/overview/what-is-paas/>

10.1.3. Benefícios do uso de PaaS

A redução de custo é o seu maior benefício, mas, há também outros benefícios, como: Agilidade, onde reduz o tempo de codificação, desenvolvimento simplificado para diversas plataformas. Fornece uma estrutura na qual os desenvolvedores podem compilar para desenvolver ou personalizar aplicativos baseados em nuvem como escalabilidade, alta disponibilidade e funcionalidades de multilocatário reduzindo a quantidade de codificação que os desenvolvedores devem fazer. As ferramentas fornecidas como serviço com PaaS permitem às organizações analisarem e minar seus dados e os provedores de PaaS podem oferecer outros serviços que aprimoram aplicativos, como fluxo de trabalho, diretório, segurança e agendamento.

Pontos principais

Atente para os tópicos a seguir. Eles devem ser estudados com muita atenção, pois representam os pontos mais importantes do capítulo.

- Diferenças entre PaaS, IaaS e SaaS;
- PaaS é um ambiente de desenvolvimento e implantação de aplicativos compartilhado.
- Benefícios do PaaS.

11. Operação de base de dados

Banco de Dados na nuvem.

- ✓ Banco de Dados na nuvem;
- ✓ Vantagens;
- ✓ Opção pela migração;

11.1. Banco de Dados na nuvem

O banco de dados na nuvem reside em uma plataforma de computação em nuvem privada, pública ou hibrida. Há dois modelos em nuvem: tradicional ou banco de dados como serviço (DBaaS). No primeiro caso o banco de dados é executado na infraestrutura de TI interna através de uma máquina virtual e o modelo DBaaS é um serviço de assinatura baseado em taxas no qual o banco de dados é executado na infraestrutura física do provedor de serviços.A empresa cliente pode configurar um arranjo de hospedagem gerenciada, no qual o provedor lida com a manutenção e gerenciamento de banco de dados. <http://www.advancedit.com.br/wp-content/uploads/2018/04/E_book_AdvancedIT-_-BANCO-DE-DADOS-NA-NUVEM-_DbaaS_.compressed.pdf>

11.1.1. Vantagens

Um banco de dados da nuvem oferece as seguintes vantagens:

- Eliminação da infra física
- Redução de custos
- Por meio de um acordo de nível de serviço (SLA), o provedor DBaaS pode fornecer garantias que geralmente quantificam a disponibilidade mínima de tempo de atividade e os tempos de resposta das transações.
- Neste tipo de ambiente geralmente as empresas oferecem aos seus clientes tecnologia de última geração, especialidade, suporte full time e entre outros diferenciais.

11.1.2. Opção pela migração

Uma empresa que deseja optar pela migração do seu Banco de Dados necessita avaliar alguns itens:

- É mais seguro que a migração seja gradativa;
- Deve-se avaliar o tamanho do Banco de Dados;
- Realizar testes antes da Migração total dos Dados.
- Escolher um provedor com um SLA que atende as necessidades da empresa. Considerando que, ao escolher a empresa deve-se avaliar a seriedade da empresa contratada pois o relacionamento será de longo prazo.
- Garantir a escalabilidade.
- Avaliar os custos envolvidos na migração

A empresa que está contratando o serviço deve compreender e gerenciar as transferências de dados e a latência. Conhecer a nuvem do provedor de serviços. Estar ciente da estrutura de custos, como: como classe de instância, tempo de execução, armazenamento principal e backup, solicitações de E/S por mês e transferência de dados – e suas expectativas de crescimento ao longo do tempo. Se preocupar com os requisitos de segurança da empresa. Em resumo deve-se monitorar e otimizar seu ambiente de nuvem. Executar o banco de dados na nuvem pode proporcionar maior flexibilidade e agilidade e menos do tempo sendo gasto com a administração do banco de dados, no entanto, a empresa não se livra das responsabilidades de gerenciamento do banco de dados. <http://www.sispro.com.br/blog/cloud-computing/o-que-e-necessario-para-gerenciar-o-banco-de-dados-na-nuvem/>

Pontos principais

Atente para os tópicos a seguir. Eles devem ser estudados com muita atenção, pois representam os pontos mais importantes do capítulo.

- O que o Banco de Dados na nuvem tem a oferecer.
- Quais são os benefícios e os cuidados com o Banco de Dados na nuvem.

Referências

A KENNETH REITZ PROJECT. **O Guia do Mochileiro para Python!** 2016. Disponível em: <http://python-guide-pt-br.readthedocs.io/pt_BR/latest/>. Acesso em: 10 abr. 2018.

ALJORD, Patrick; CHACON, Scott. **Pro Git.** Estados Unidos: Apress, 2009. 300 p

AMAZON. **O que é o controle do código-fonte?** Disponível em: <https://aws.amazon.com/pt/devops/source-control/>. Acesso em: 09 abr. 2018

FOWLER, Martin. **Continuous Integration.** 2006. Disponível em: <https://martinfowler.com/articles/continuousIntegration.html>. Acesso em: 09 abr. 2018.

IZABEL, Leonardo Roxo Pessanha. **TESTES AUTOMATIZADOS NO PROCESSO DE DESENVOLVIMENTO DE SOFTWARES.** 2014. 65 f. Monografia (Especialização) - Curso de Engenharia Eletrônica e de Computação, Escola Politécnica, Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2014. Cap. 3. Disponível em: <http://www.monografias.poli.ufrj.br/monografias/monopoli10012548.pdf>. Acesso em: 10 abr. 2018.

A KENNETH REITZ PROJECT. **O Guia do Mochileiro para Python!** 2016. Disponível em: <http://python-guide-pt-br.readthedocs.io/pt_BR/latest/>. Acesso em: 10 abr. 2018.