DEVOPS

Marcel Santos Silva

SAGAH
SOLUÇÕES EDUCACIONAIS INTEGRADAS

Tipos de controles de versão e verbetes

Objetivos de aprendizagem

Ao final deste texto, você deve apresentar os seguintes aprendizados:

- Listar os sistemas mais utilizados de controles de versão de código-fonte.
- Demonstrar as principais operações utilizadas nos sistemas de versionamento.
- Diferenciar sistemas de versionamento distribuídos de sistemas de versionamento centralizados.

Introdução

Cada vez mais há necessidade de agilidade no fluxo de trabalho de desenvolvimento de *software*. Em virtude da enorme quantidade de linhas de código e dos inúmeros projetos em desenvolvimento, tornou-se comum a utilização de ferramentas e padrões que viabilizam um maior controle no ambiente DevOps. Para Kim *et al.* (2018), no DevOps ideal, os programadores recebem *feedback* rápido e frequente com relação ao seu trabalho, o que possibilita que eles possam implementar, integrar e validar o seu código-fonte com maior rapidez. Mas como é possível controlar todo um processo de desenvolvimento de *software* sem medo de perder informações ou sobrescrever atualizações? A resposta para essa questão é a utilização de um controle de versão.

Um sistema de controle de versão, também conhecido como VCS (do inglês *Version Control System*) ou SCM (do inglês *Source Code Management*) na função prática da computação e da engenharia de *software*, tem como finalidade gerenciar várias versões no desenvolvimento de um documento qualquer. Isso garante a gestão de forma mais inteligente e eficaz para organizar qualquer projeto, possibilitando o acompanhamento de históricos de desenvolvimento, atividades paralelas, customização de uma determinada versão e finalidades específicas sem a necessidade de

se alterar o projeto principal ou até mesmo recuperar uma versão anterior, caso tenha ocorrido perda de alguma informação.

Os sistemas de controle de versão são muito utilizados no desenvolvimento de sistemas pelos iniciantes na área, que, ao serem inseridos em uma equipe de programadores, não conhecem as ferramentas utilizadas para que se consiga trabalhar em paralelo de forma eficiente, excluindo-se as possibilidades de sobreposição de alterações, utilização de versão errada, entre outras.

Neste capítulo, você conhecerá os principais tipos de controle de versão, também conhecido como versionamento. Além disso, verá quais são as principais operações utilizadas nos sistemas de versionamento. Por fim, conhecerá a diferença entre sistemas de versionamento distribuídos e centralizados.

1 Controle de versão

De maneira bem simples, os arquivos de um projeto são armazenados em um repositório, no qual o histórico de suas versões é registrado. Os programadores podem acessar e recuperar a última versão disponível e realizar uma cópia local, que possibilitará fazer alterações sobre ela. É possível submeter cada alteração executada ao repositório e recuperar as atualizações feitas pelos outros membros da equipe.

Dentro dessa arquitetura, é importante ressaltar que o controle de versão suporta o desenvolvimento em algumas formas, apresentadas a seguir:

- **Registrando o histórico:** armazena toda a evolução do projeto, ou seja, toda alteração realizada é registrada no repositório, o que possibilita a identificação de autor, data e origem das alterações. Uma característica importante é a possibilidade de reconstrução de determinada revisão específica do código-fonte, sempre que necessário.
- **Colaborando concorrentemente:** permite que mais de um programador realize alterações em paralelo sobre um mesmo código-fonte, sem sobrescrever as modificações de outro membro da equipe.
- **Variações no projeto:** proporciona a manutenção de versões diferentes de evolução do mesmo projeto. Ou seja, a versão 1.0 é a oficial, enquanto se prepara a versão 2.0.

Os controles de versões são compostos, basicamente, por duas partes: o **repositório** (servidor), que armazena todo o histórico de ajustes do projeto, registrando todas as alterações realizadas nos itens versionados; e a **estação de trabalho**, que possui uma cópia dos arquivos do projeto vinculada ao servidor para a identificação de possíveis modificações (Figura 1). Cada estação de trabalho é considerada individual e isolada das demais.

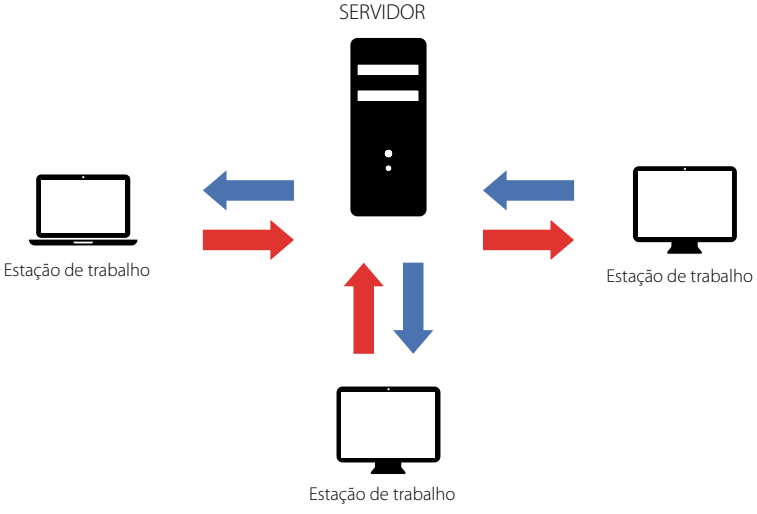

Figura 1. Arquitetura básica do controle de versão.

O processo de sincronização entre a estação de trabalho e o repositório é realizado por meio dos comandos `commit` e `update`. O comando `commit` submete um pacote com as modificações feitas pela estação de trabalho (origem) ao repositório (destino). Já o comando `update` realiza o processo inverso, ou seja, disponibiliza as alterações submetidas pelas demais estações de trabalho ao repositório (origem) para a estação de trabalho (destino), que deseja atualizar o projeto.

É importante ressaltar que todo `commit` cria uma revisão no servidor contendo as alterações, a data e o usuário responsável. O conceito de **revisão** pode ser ilustrado como uma fotografia de todos os arquivos e diretórios em um momento específico do projeto. Os registros antigos são guardados e podem ser recuperados assim que houver necessidade, e o conjunto dessas revisões é especificamente o histórico de alterações do projeto.

Existem diversas ferramentas para controle de versão no mercado, sendo algumas gratuitas e outras proprietárias. A seguir, serão apresentados os principais sistemas utilizados pelas equipes de desenvolvimento de *software* para controle de versionamento.

GIT

A ferramenta Git (Figura 2) é um dos sistemas de controle de versão e de controle de código-fonte distribuído mais conhecidos e utilizados pelas equipes de desenvolvimento de *software*. O Git foi criado por Linus Torvalds especificamente para o desenvolvimento do *kernel* Linux, porém tornou-se popular e foi adotado para outros projetos.

Figura 2. Logotipo do Git.
Fonte: Git (2020, documento *on-line*).

Uma das vantagens do Git é o fato de ele ser livre e de código aberto, além de ter sido projetado para lidar com todo tipo de projeto, desde pequenos a enormes, com eficiência e rapidez. Algumas características interessantes do Git são a garantia de dados, área de teste, ramificação e fusão.

Para o gerenciamento dos repositórios na *web*, os desenvolvedores utilizam, em sua maioria, o *github* ou o *bitbucket*.

Redmine

A ferramenta Redmine (Figura 3) consiste em uma aplicação *web* desenvolvida por meio do uso do *framework* Ruby on Rails, além de ser multiplataforma e utilizar um banco de dados cruzado.

Figura 3. Logotipo do Redmine.
Fonte: Redmine (2020, documento *on-line*).

Dentre as características do Redmine, destacam-se as apresentadas a seguir:

- suporte para múltiplos projetos;
- controle flexível de acesso baseado em função;
- sistema de rastreamento de problemas flexíveis;
- rastreamento do tempo;
- suporte a vários bancos de dados;
- integração com diversos repositórios (SVN, CVS, Git, Mercurial e Bazaar).

IBM Rational ClearCase

O IBM Rational ClearCase (Figura 4) disponibiliza acesso controlado aos ativos de *software*, incluindo código, requisitos, documentos de projeto, modelos, planos e resultados de testes. Além disso, ele oferece suporte a desenvolvimento paralelo, gerenciamento automatizado do espaço de trabalho, gerenciamento de linha de base, gerenciamento seguro de versões, auditoria confiável de compilação e acesso flexível.

Figura 4. Logotipo do IBM Rational ClearCase.
Fonte: IBM (2020, documento *on-line*).

Dentre as características do IBM Rational ClearCase, destacam-se as apresentadas a seguir:

- gerenciamento automatizado do espaço de trabalho;
- acesso transparente e em tempo real a arquivos e diretórios;
- suporte a desenvolvimento paralelo;
- gerenciamento avançado de criação e lançamento;
- acesso local, remoto e de cliente à *web*.

Subversion

O Subversion (Figura 5) é um sistema de controle de versão de código aberto, centralizado e caracterizado, em virtude de sua confiabilidade, como um repositório seguro para projetos e dados importantes. Alguns pontos de destaque são: a simplicidade do modelo e seu fácil uso e a capacidade de suportar as demandas de uma variedade de usuários e projetos para operações corporativas de grande escala.

Figura 5. Logotipo do Subversion.
Fonte: IBM (2020, documento *on-line*).

Dentre as características do Subversion, destacam-se as apresentadas a seguir:

- versionamento de diretórios;
- registro de cópias, exclusões e renomeação;
- metadados versionados de forma livre;
- *commits* únicos e individuais;
- bloqueio de arquivos;
- resolução de conflitos interativos.

Mercurial

O Mercurial (Figura 6) é um sistema de controle de versão distribuído e gratuito que gerencia de forma eficaz projetos de qualquer tamanho, além de oferecer uma interface fácil e intuitiva. Cada cópia possui todo o histórico do projeto, com isso, a maioria das ações é realizada localmente, de forma rápida e conveniente.

Figura 6. Logotipo do Mercurial.
Fonte: Mercurial (2020, documento *on-line*).

Dentre as características do Mercurial, destacam-se as apresentadas a seguir:

- suporte a vários fluxos de trabalho;
- arquitetura distribuída;
- facilidade de uso;
- extensível;
- código aberto;
- plataforma independente.

Os principais objetivos do desenvolvimento do Mercurial incluem alta *performance* e escalabilidade, descentralização, desenvolvimento colaborativo distribuído, controle de arquivos textuais e binários de forma robusta e operações avançadas de ramos e mesclagem. Além disso, o Mercurial inclui, de forma integrada, um sistema de visualização dos repositórios via *web* e facilitação na transição de usuários do Subversion.

Darcs

O Darcs (Figura 7) é um sistema de controle de versão de código aberto e gratuito e de plataforma cruzada, como Git, Mercurial ou Subversion. No entanto, uma característica diferenciada é o seu foco em mudanças, em vez de em *snapshots*. O Darcs permite, ainda, uma forma de trabalho mais livre e interface de usuário simplista. Outro ponto a ser considerado como diferencial é que ele não exige um servidor centralizado, funcionando perfeitamente no modo *off-line*.

Figura 7. Logotipo do Darcs.
Fonte: Darcs (2020, documento *on-line*).

Dentre as características do Darcs, destacam-se as apresentadas a seguir:

- modo *off-line*;
- preparação local;
- fácil ramificação e fusão;
- fácil colaboração por *e-mail*;
- desenvolvimento paralelo;
- interatividade;
- hospedagem própria.

BAZAAR

O Bazaar (Figura 8) é um sistema de controle de versão que tem a capacidade de rastrear o histórico do projeto ao longo do tempo, além de possibilitar a colaboração entre desenvolvedores.

Figura 8. Logotipo do Bazaar.
Fonte: Bazaar (2020, documento *on-line*).

Dentre as características do Bazaar, destacam-se as apresentadas a seguir:

- trabalho *off-line*;
- suporte a diversos fluxos de trabalho;
- suporte multiplataforma;
- rastreamento de renomeação e mesclagem inteligente;
- alta eficiência e velocidade de armazenamento.

O Git e o Mercurial são muito flexíveis em relação aos fluxos de trabalho, porém o Bazaar é a única dessa ferramenta que suporta ramificações vinculadas, sendo uma forma mais fácil e segura de implementar um fluxo de trabalho centralizado.

2 Operações em controle de versões

Para melhor compreensão do funcionamento dos sistemas de controle de versão, é importante conhecer as principais operações que os envolvem. A seguir, são apresentados os elementos mais utilizados pela maioria dos controles de versionamento:

- ***Commit* (*Checkin*):** criação de uma nova versão do projeto.
- ***Checkout*:** recuperação de uma versão específica do projeto ou arquivo.
- ***Revert*:** possibilita ao desenvolvedor descartar as mudanças realizadas em estação local, recuperando a mesma versão do repositório.
- ***Diff*:** garante a possibilidade de comparação do arquivo na estação local com qualquer outra versão do repositório.
- ***Delete*:** permite a exclusão de um arquivo do repositório. Quando as demais estações de trabalho realizarem um *update*, o arquivo será efetivamente excluído do repositório.
- ***Lock*:** possibilita o travamento de determinado arquivo, de forma que nenhum outro usuário o modifique.

Saiba mais

Uma ***tag*** é simplesmente uma representação mais simples de se lembrar, para seres humanos, de uma determinada versão do código.

Confira, a seguir, alguns conceitos relevantes com relação aos controles de versões.

- **Repositório de versões:** armazenamento de todas as versões dos arquivos sob o controle versões.
- **Repositório de versões centralizado:** cada estação de trabalho local contém apenas uma versão específica da árvore de versões do repositório central, e todos os usuários realizam as operações de controle de versões nesse repositório.
- **Repositório de versões distribuído:** cada estação local possui um repositório acoplado, de modo que o usuário possui um repositório próprio para realizar o controle de versões. As operações realizadas sobre os arquivos

são feitas no repositório local do usuário, e operações específicas dos repositórios distribuídos são utilizadas para sincronizar repositórios diferentes.

- **Árvore de revisões ou de versões:** estrutura lógica que mapeia todas as versões armazenadas no repositório para determinado arquivo ou conjunto de arquivos.

Fique atento

É importante ressaltar que o repositório evita a perda de dados, porém o que está na estação local não pode ser recuperado se houver um problema com o computador. É possível, ainda, ignorar arquivos para que eles não sejam comitados nem apareçam na lista de arquivos passíveis de *commit*.

Para cada um dos tipos de controle de versão, há um conjunto de operações básicas, conforme o Quadro 1, a seguir.

Quadro 1. Operações básicas dos controles de versão centralizado e distribuído

Descrição	Controle de versão centralizado	Controle de versão distribuído
Criação de estação de trabalho ou repositório.	*CHECKOUT*	*CLONE*
Envio de modificações para o repositório, gerando uma revisão.	*COMMIT*	*COMMIT*
Alteração da estação de trabalho em uma revisão.	*UPDATE*	*UPDATE*
Importação de revisões feitas em outro repositório.		*PULL*
Envio de revisões locais para outro repositório.		*PUSH*

Uma característica importante dos controles de versão é a possibilidade de separar modificações em um caminho diferente para cada desenvolvimento, conforme a Figura 9. Esse caminho, conhecido como ***branch*** (seta verde), é utilizado especialmente para a implementação de novas funcionalidades, sem comprometer o caminho principal da implementação, denominado ***trunk***, (seta azul), com erros de compilação e *bugs*. A *branch* só será integrada ao *trunk* quando ela se tornar estável. Há também a possibilidade de congelamento de revisão, denominada ***tag*** (seta cor de laranja), isto é, é um estado fixo do produto que possui um conjunto de funcionalidades estáveis que não sofrerão mais nenhuma alteração.

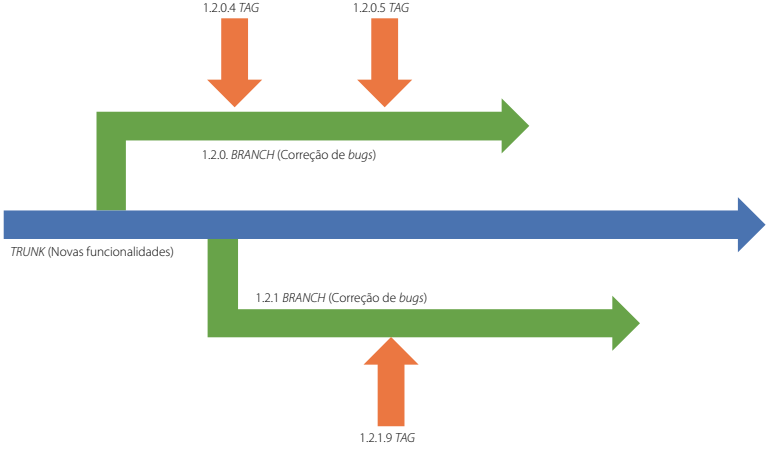

Figura 9. Diferenças entre *trunk*, *branch* e *tag*.
Fonte: Adaptada de Guelfi (2015).

3 Versionamentos distribuído e centralizado

Existem dois tipos de sistemas de controles de versão: centralizado e distribuído. Ambos possuem repositórios e estações de trabalho, porém a diferença entre eles está em como cada um está estruturado e organizado. A seguir, são apresentados os detalhes e as diferenças entre esses dois tipos de versionamento.

Versionamento centralizado

O sistema de controle de versão centralizado é composto por um único servidor central e várias estações de trabalho, com base no conceito de arquitetura cliente–servidor. Por ser um sistema centralizado, as estações de trabalho necessitam consultar o servidor para a realização da comunicação. Esse modelo atende à maioria das equipes de desenvolvimento de sistemas de médio porte para a realização de implementação por meio de uma rede local, além de não necessitar de velocidade para o envio e o recebimento dos dados. Um dos sistemas mais comuns com esse tipo de controle de versão centralizado é o Subversion.

Os sistemas de controle de versão centralizados possuem uma topologia em forma de estrela, ou seja, há um único repositório central com diversas estações de trabalho, uma para cada desenvolvedor. A comunicação entre as estações de trabalho passa, obrigatoriamente, pelo repositório central, conforme a Figura 10.

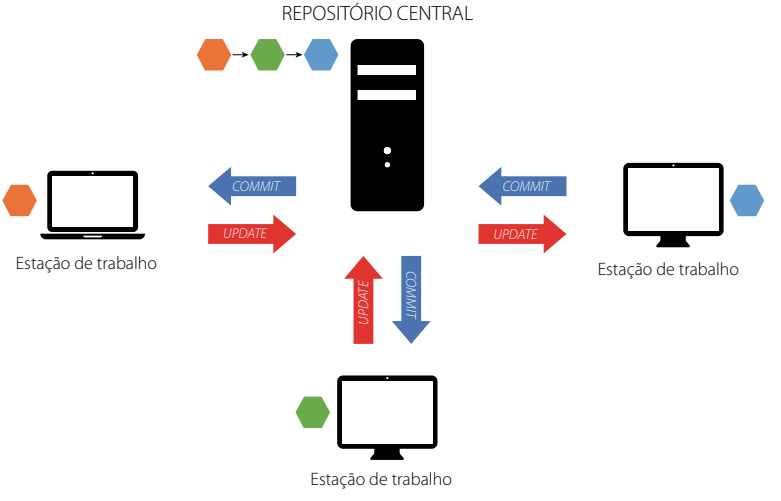

Figura 10. Sistema de controle de versão centralizado.

Algumas vantagens do uso do versionamento centralizado são: maior controle do projeto, imposição de segurança de acesso com facilidade e possibilidade de bloqueio de arquivos específicos, sendo ideal para equipes pequenas. Com relação às desvantagens, pode-se considerar: baixa escalabilidade e necessidade constante de conexão com a internet.

Versionamento distribuído

Os sistemas de controle de versão distribuídos podem ser definidos como diversos repositórios autônomos e independentes, um para cada programador. Cada repositório tem a sua estação de trabalho acoplada, onde as operações *commit* e *update* ocorrem localmente, conforme a Figura 11.

Essa arquitetura é recomendada para equipes com uma grande quantidade de desenvolvedores que estão remotamente distantes. O funcionamento do sistema de controle de versão distribuído ocorre da seguinte forma: cada estação de trabalho possui o seu próprio repositório, ou seja, as operações de *checkin* e *checkout* são realizadas de forma local. Ao contrário da arquitetura centralizada, essas estações de trabalho podem se comunicar entre si, porém é recomendável que se utilize um servidor responsável pelo envio dos arquivos para que se organize o fluxo e se evite ramificações do projeto e a perda do controle. Na maioria das vezes, o sistema oferece um servidor remoto para que o projeto seja hospedado. O processo de comunicação entre o servidor principal e as estações de trabalho funciona por meio de duas operações: uma para atualizar (puxar) e outra para mesclar o projeto (empurrar), conhecidas como ***pull*** e ***push***, respectivamente.

Considerando-se que o processo é local, o sistema de controle distribuído possui maior rapidez, porém exige um maior conhecimento da ferramenta e, inicialmente, pode confundir o programador. Por exemplo, o sistema de mesclagem em alterações concorrentes torna-se distinto por trabalhar em um sistema de arquivos binários, que, em determinadas situações, não possibilita a comparação entre essas atualizações ao mesmo tempo. Já os sistemas de controle de versão centralizados utilizam arquivos de texto, o que permite a comparação em alterações concorrentes, apresentando ao programador a possibilidade de escolher a solução ideal.

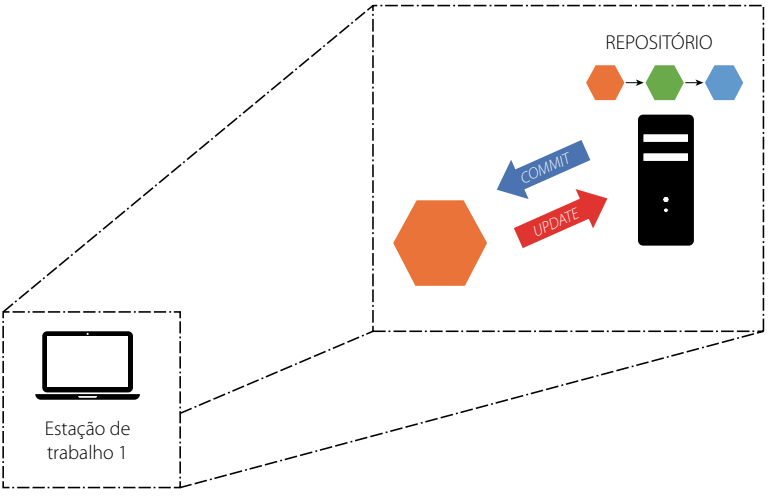

Figura 11. Sistema de controle de versão distribuído.

Com relação ao sistema de controle de versão distribuído, algumas das vantagens do seu uso são replicação do repositório, maior rapidez e autonomia, pois ele permite a realização de alteração *off-line*. No que se refere às desvantagens, pode-se citar a complexidade do fluxo de trabalho e a dificuldade em bloqueio de arquivos específicos.

Por fim, observa-se que os sistemas de controle de versão resolvem muitos problemas relacionados diretamente ao desenvolvimento de *software*. Atualmente, é prática comum a utilização dessa forma de trabalho, e existem inúmeras ferramentas disponíveis no mercado, conforme apresentado. É importante ressaltar que, antes de escolher qual sistema utilizar, deve-se analisar as opções e identificar a solução que melhor atende às necessidades da equipe de desenvolvimento.

Referências

BAZAAR. [*Site*]. [2020]. Disponível em: https://bazaar.canonical.com/en/. Acesso em: 26 ago. 2020.

DARCS. [*Site*]. [2020]. Disponível em: http://darcs.net/. Acesso em: 26 ago. 2020.

GIT. [*Site*]. [2020]. Disponível em: https://git-scm.com/. Acesso em: 26 ago. 2020.

GUELFI, E. *Conceitos do controle de versão*: criando branchs e tags utilizando Tortoise SVN. 2015. Disponível em: https://tsdn.tecnospeed.com.br/blog-do-desenvolvimento-tecnospeed/post/conceitos-do-controle-de-versao-criando-branchs-e-tags-utilizando-tortoise-svn#:~:text=Logo%20de%20cara%20podemos%20dizer,funcionalidades%20que%20n%C3%A3o%20ser%C3%A3o%20mais. Acesso em: 26 ago. 2020.

IBM. *What can IBM Rational ClearCase do for my business?* [2020]. Disponível em: https://www.ibm.com/us-en/marketplace/rational-clearcase. Acesso em: 26 ago. 2020.

KIM, G. *et al. Manual de DevOps*: como obter agilidade, confiabilidade e segurança em organizações tecnológicas. Rio de Janeiro: Alta Books, 2018.

MERCURIAL. [*Site*]. [2020]. Disponível em: https://www.mercurial-scm.org/. Acesso em: 26 ago. 2020.

REDMINE. [*Site*]. [2020]. Disponível em: https://www.redmine.org/projects/redmine/wiki/Logo. Acesso em: 26 ago. 2020.

Leituras recomendadas

ARUNDEL, J.; DOMINGUS, J. *DevOps nativo de nuvem com Kubernetes*: como construir, implantar e escalar aplicações modernas na nuvem. São Paulo: Novatec, 2019.

MUNIZ, A. *et al. Jornada DevOps*: unindo cultura ágil, Lean e tecnologia para entregar software com qualidade. 2. ed. Rio de Janeiro: Brasport, 2020.

Fique atento

Os *links* para *sites* da *web* fornecidos neste capítulo foram todos testados, e seu funcionamento foi comprovado no momento da publicação do material. No entanto, a rede é extremamente dinâmica; suas páginas estão constantemente mudando de local e conteúdo. Assim, os editores declaram não ter qualquer responsabilidade sobre qualidade, precisão ou integralidade das informações referidas em tais *links*.

Conteúdo:

SAGAH
SOLUÇÕES EDUCACIONAIS INTEGRADAS