



---

# PRÁTICAS DA CULTURA DEVOPS NO DESENVOLVIMENTO DE SISTEMAS

**Anderson Pereira de Lima Jerônimo**

# **PRÁTICAS DA CULTURA DEVOPS NO DESENVOLVIMENTO DE SISTEMAS**

**1<sup>a</sup> edição**

São Paulo  
Platos Soluções Educacionais S.A  
2021

**© 2021 por Platos Soluções Educacionais S.A.**

Todos os direitos reservados. Nenhuma parte desta publicação poderá ser reproduzida ou transmitida de qualquer modo ou por qualquer outro meio, eletrônico ou mecânico, incluindo fotocópia, gravação ou qualquer outro tipo de sistema de armazenamento e transmissão de informação, sem prévia autorização, por escrito, da Platos Soluções Educacionais S.A.

**Head de Platos Soluções Educacionais S.A**

Silvia Rodrigues Cima Bizatto

**Conselho Acadêmico**

Carlos Roberto Pagani Junior  
 Camila Turchetti Bacan Gabiatti  
 Camila Braga de Oliveira Higa  
 Giani Vendramel de Oliveira  
 Gislaine Denisale Ferreira  
 Henrique Salustiano Silva  
 Mariana Gerardi Mello  
 Nirse Ruscheinsky Breternitz  
 Priscila Pereira Silva  
 Tayra Carolina Nascimento Aleixo

**Coordenador**

Henrique Salustiano Silva

**Revisor**

Stella Marys Dornelas Lamounier

**Editorial**

Alessandra Cristina Fahl  
 Beatriz Meloni Montefusco  
 Carolina Yaly  
 Mariana de Campos Barroso  
 Paola Andressa Machado Leal

---

Dados Internacionais de Catalogação na Publicação (CIP)

J56p Jerônimo, Anderson Pereira de Lima  
 Práticas da cultura DevOps no desenvolvimento de  
 Sistemas / Anderson Pereira de Lima Jerônimo. – São  
 Paulo: Platos Soluções Educacionais S.A., 2021.  
 44 p.

ISBN 978-65-5356-056-7

1. Cultura devops. 2. Desenvolvedores full stack.  
 3. Desenvolvimento web. I. Título.

CDD 004.678

---

Evelyn Moraes – CRB-8 010289

2021

Platos Soluções Educacionais S.A  
 Alameda Santos, nº 960 – Cerqueira César  
 CEP: 01418-002— São Paulo — SP  
 Homepage: <https://www.platosedu.com.br/>

# PRÁTICAS DA CULTURA DEVOPS NO DESENVOLVIMENTO DE SISTEMAS

## SUMÁRIO

Conceitos Essenciais sobre a Cultura DevOps aplicada ao Desenvolvimento de Sistemas	05
Selecionando e conhecendo ferramentas do DevOps no Desenvolvimento Web	18
Tecnologia de versionamento GIT e de integração continua	30
Adoção das Práticas DevOps em Equipes de Desenvolvimento	43

# **Conceitos Essenciais sobre a Cultura DevOps aplicada ao Desenvolvimento de Sistemas.**

Autoria: Anderson Pereira de Lima Jerônimo

Leitura crítica: Stella Marys Dornelas Lamounier



## **Objetivos**

- Conhecer sobre a cultura DevOps.
- Entender a importância da Metodologia Ágil para DevOps
- Compreender a integração das equipes de Tecnologia da Informação com a equipe de Desenvolvimento.



## 1. Cultura DevOps

O movimento cultural chamado DevOps surgiu da necessidade de obter maior colaboração entre as equipes de desenvolvimento e as equipes de operações. Um dos problemas enfrentados entre as equipes é questão do isolamento de suas tarefas e atividades específicas, pela falta de comunicação mais alinhada que prejudica entregas do software em tempo hábil.

A palavra DevOps é união de dois termos em inglês que identificam as equipes envolvidas nas atividades de criação e publicação do software (MUNIZ, SANTOS, *et al.*, 2019):

*Development* (desenvolvimento): equipe responsável pela identificação dos requisitos com o cliente, análise, testes e toda parte de codificação.

*Operations* (operações): equipe responsável pela publicação (*deploy*) em produção, pelo monitoramento de falhas e problemas.

Nesse contexto, o DevOps surgiu como uma vertente da metodologia ágil, que caracteriza com variedade de ferramentas, e dinâmicas, com a finalidade de minimizar os obstáculos enfrentados entre as equipes de tecnologia da informação e seus desenvolvedores.

**Figura 1 – Equipe de tecnologia**



Fonte: scyther5/ iStock.com.

Podemos descrever o DevOpsr como cultura uma cultura baseada em modelos, ferramentas e, principalmente, em práticas, com o foco de permitir essa integração mais fluida entre os profissionais na área de tecnologia da informação, atuando cada vez mais em demandas por sistemas complexos exigidos pelo mercado.

Essa nova abordagem orienta as equipes a trabalharem com propósito de otimizar tarefas com objetivo de um resultado mais alinhado e com adaptação contínua dos softwares. Com surgimento dessa cultura, os profissionais de diferentes áreas de TI se sentem mais confiantes em adotar parâmetros de padronização de interfaces e simplificação de seus modelos.

Podemos destacar que o DevOps se opõe ao modelo tradicional, que depende de um processo em cascata, no qual só se pode iniciar uma etapa quando outra é encerrada (SOMMERVILLE, 2019).

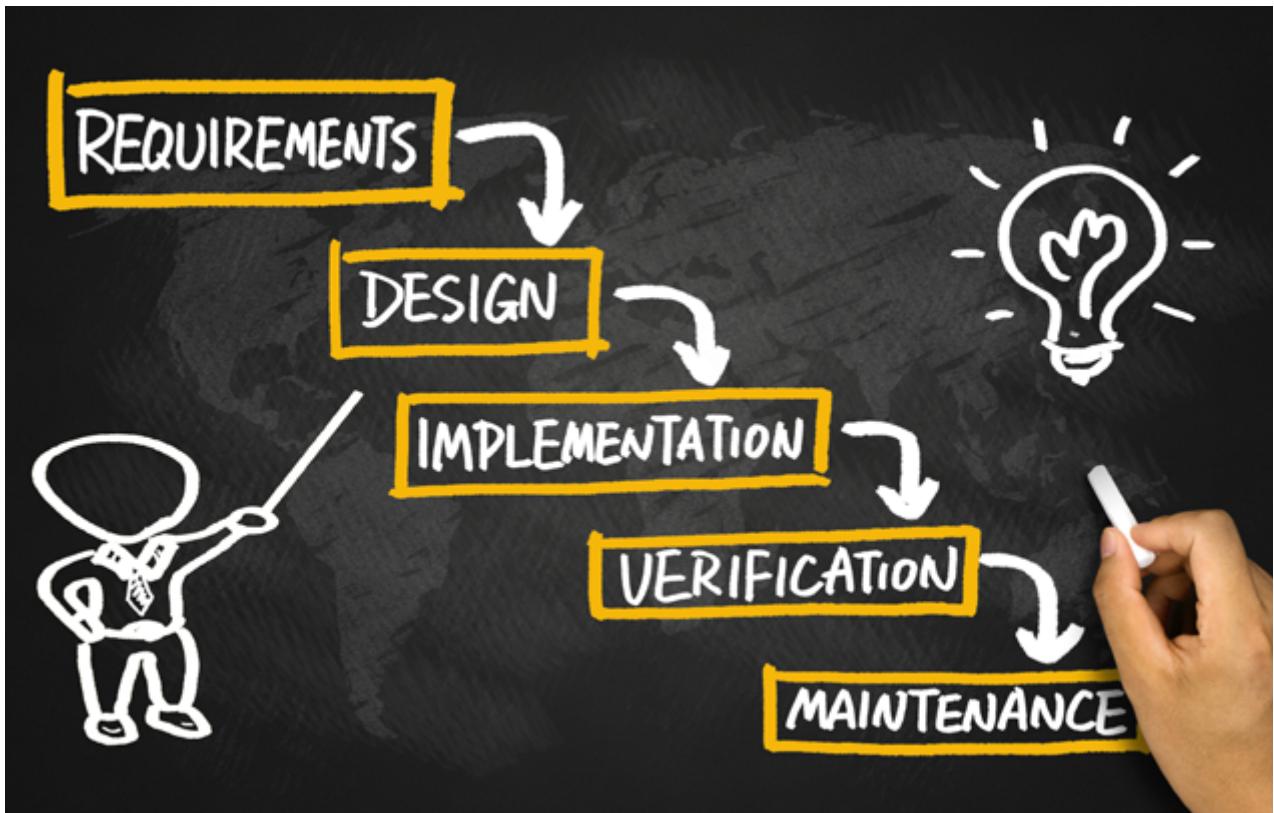
O modelo tradicional é baseado em processos orientados a documentação, que, de certa maneira, impõem entraves no desenvolvimento de software que exija mais dinamismo e entregas constantes no decorrer do projeto, acarretando uma enorme chance de impactar de forma negativa a entrega de releases e, com isso, aumento nos custos desenvolvimento.

Apesar do modelo em cascata ser usado para desenvolvimento de software, é mais indicado para desenvolvimento de hardware, pelo fato desses projetos terem altos custos de produção, onde cada ciclo precisa ser terminado e aprovada pelo gerente de projeto, para iniciar a próxima fase.

Vale salientar que o modelo em cascata deve ser aplicado no desenvolvimento de software quando houver requisitos do sistema estáveis e os requisitos futuros já presumíveis (SOMMERVILLE, 2019).

Este modelo é composto por algumas as fases: requisitos, design, implementação, verificação e manutenção. Sendo que a fase de requisitos é a mais importante para todo ciclo, pois, por meio dela, a equipe de desenvolvimento, seja de hardware e software, estabelece um plano dirigido a ser seguido.

Figura 2 – Modelo cascata



Fonte: cacaroot/ iStock.com.

Entretanto, há a necessidade de criação de aplicações mais dinâmicas que exigem mudanças rápidas de acordo com as demandas do mercado. Houve um movimento chamado de manifesto ágil, que ocorreu em 2001, que trata de um documento que propõe uso de métodos ágeis como forma oficial. (FERREIRA, 2020). No manifesto ágil, fica destacado que os indivíduos e intenções são mais importantes do que processos e ferramentas.

Nesse escopo, as entregas funcionais são mais eficazes do que ter uma documentação mais abrangente e a colaboração com cliente é mais importante que negociação de contrato, estreitando uma resposta rápida a mudanças importantes no decorrer do projeto.

Nos métodos ágeis propostos pelo manifesto, temos uma nova abordagem com relação ao modelo tradicional, pelo fato das etapas do processo ocorrerem em paralelo o tempo todo, sendo com interações

curtas. Ao final de cada interação, o processo de software tende a agregar mais funcionalidades com menos bugs, e a equipe decide de forma colaborativa com cliente qual será a próxima demanda a ser desenvolvida, de acordo com as prioridades do projeto.

**Figura 3 – Manifesto ágil**



Fonte: akinbostanci/ iStock.com.

Nesse contexto, a cultura DevOps ganhou mais força em 2009, quando a demanda por aplicativos e desenvolvimento de software cresce de forma mais constante, demanda de equipes interdisciplinares, com a chegada de smartphones e *smart devices*.

## 1.1 Finalidade do DevOps

O DevOps trata da união das equipes para aceleração do processo de software mais coeso. Quando não há essa junção, a tendência é que as equipes venham a trabalhar de forma separada, o que aumenta a

probabilidade de ocorrer o surgimento de falhas, atrasos e eficiência dos softwares.

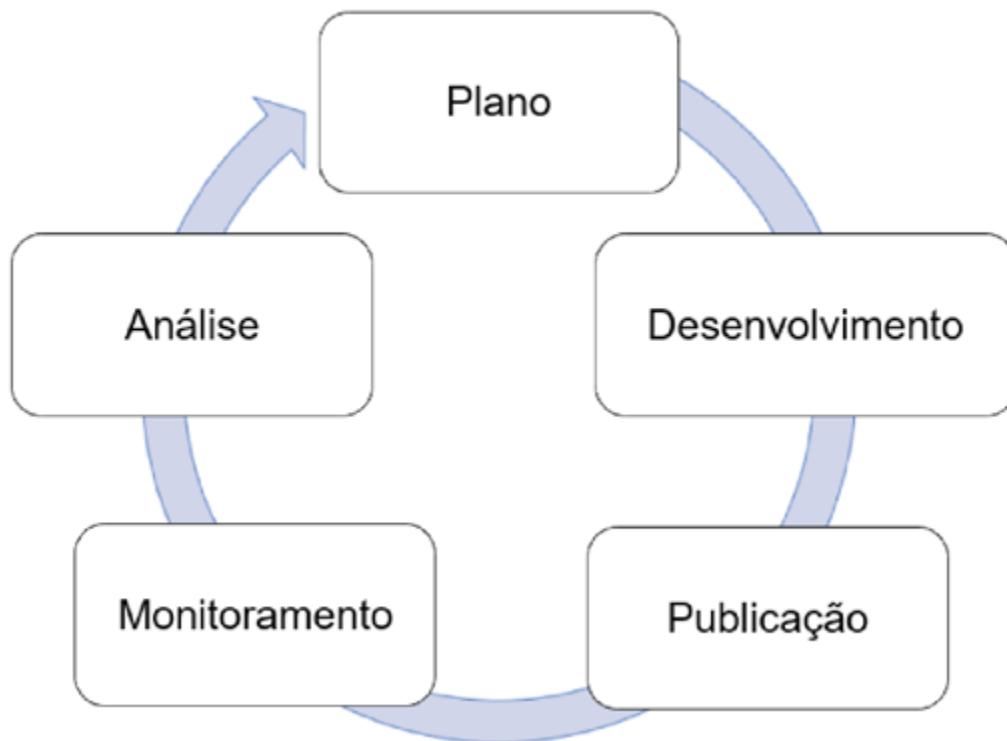
Como a proposta da metodologia ágil é de que haja uma equipe multidisciplinar, no qual os processos venham ocorrer de forma paralela, a cultura DevOps se encaixa muito bem (MUNIZ, *et al.*, 2019), isto é, possibilita o atendimento às demandas do mercado, de forma cada vez mais rápida, trazendo atualizações com novas funcionalidades e robustez de forma mais frequente.

Devido ao ambiente colaborativo entre as equipes, esta cultura torna-se essencial para acelerar a lógica digital das empresas, fazendo acompanhar as tendências do mercado.

## 1.2 Como trabalha o DevOps?

O DevOps funciona de forma colaborativa, é importante definir o compartilhamento de responsabilidades, processos e métricas e, para tal, alcançar um desenvolvimento de softwares com mais qualidade e funcionalidades. Entretanto, o trabalho só dará o resultado esperado, se os *stakeholders* fizerem seu papel de monitorar e automatizar tarefas para construção desses sistemas, empregando uma comunicação fluida nas atividades propostas. Isso permite que empresas tenham interesse em adotar a cultura DevOps entre seus profissionais de tecnologia da informação.

**Figura 4 – Etapas do DevOps**



Fonte: elaborada pelo autor.

Ao adotar a cultura DevOps, podemos mudar o pensamento de transformação digital da empresa de forma positiva, devido sua flexibilidade, pelo fato de não haver um guia pré-estabelecido. Há uma adaptabilidade de acordo com cada modelo de negócio, que garante a livre escolha de uma metodologia ou *frameworks* específicos que venham acelerar o processo de releases.

A medida da adesão das equipes trabalharem com a cultura do DevOps, permite minimizar conflitos, conseguindo agilizar processos e alinhar as demandas, ou seja, trazendo melhor entendimento entre ambas.

Esse fluxo de atividades compartilhadas, em uma equipe multidisciplinar, faz com que os erros sejam reduzidos, trazendo consigo uma economia de tempo de trabalho.

**Figura 5 – Trabalho digital**



Fonte: metamorworks/ iStock.com.

### **1.3 Como implantar o DevOps?**

Há fatores que auxiliam nesse processo de implantação da cultura DevOps, como:

- Usar a tecnologia de forma assertiva.
- Reuniões colaborativas.
- Ambientes dinâmicos.

#### **O uso assertivo da tecnologia.**

Envolve a identificação e automatização das demandas repetitivas que podem ser feitas por sistemas, auxiliando os profissionais de TI a

se concentrarem em demandas mais complexas e que tenham mais relevância ao software.

## **Reuniões colaborativas.**

Integrar as equipes de desenvolvimento e operação, por meio de reuniões em conjunto, mantendo as equipes informadas sobre metas que deverão ser alcançadas.

## **Ambientes dinâmicos.**

É importante dispor de uma infraestrutura ideal, para que haja um ambiente sempre disponível e, quando necessário, permitir que a equipe tenha processos automatizados.

Na cultura do DevOps, algumas práticas essenciais de outros modelos ágeis, foram incluídos. Podemos citar popularmente conhecido XP (*eXtreme Programming*), que tornou seu uso indispensável para as equipes de desenvolvimento e operadores, com as seguintes práticas:

- Integração contínua.
- Entrega contínua.
- Implantação contínua.

Após as equipes estarem coesas e alinhadas, o DevOps proporciona alguns benefícios:

- Velocidade: adaptação acelerada ao mercado dinâmico.
- Entrega rápida: poder inovar mais rapidamente.
- Confiabilidade: promover uma performance estável.
- Escala: gerenciar de forma mais eficiente.

- Segurança: controles e técnicas de gerenciamento de configuração.

Além das práticas e dos benefícios citados acima, mencionaremos duas ferramentas importantes para as equipes dos DevOps:

### 1.3.1 GIT

O Git é um sistema de versionamento que ajuda as equipes gerenciarem seu trabalho em modo off-line ou remoto. Isso torna interessante, pois permite às equipes trabalharem de forma colaborativa e simultânea, possibilitando que um arquivo possa ser criado e/ou editado por mais de um membro da equipe ao mesmo tempo, registrando cada alteração local seja destinada para um único servidor remoto, que ficará responsável em controlar as versões mais atualizadas.

Esse processo garante um histórico do software para equipe, uma evolução de como o software está sendo construído, e o monitoramento que garantirá o surgimento de novas releases do projeto.

**Figura 6 – GIT**



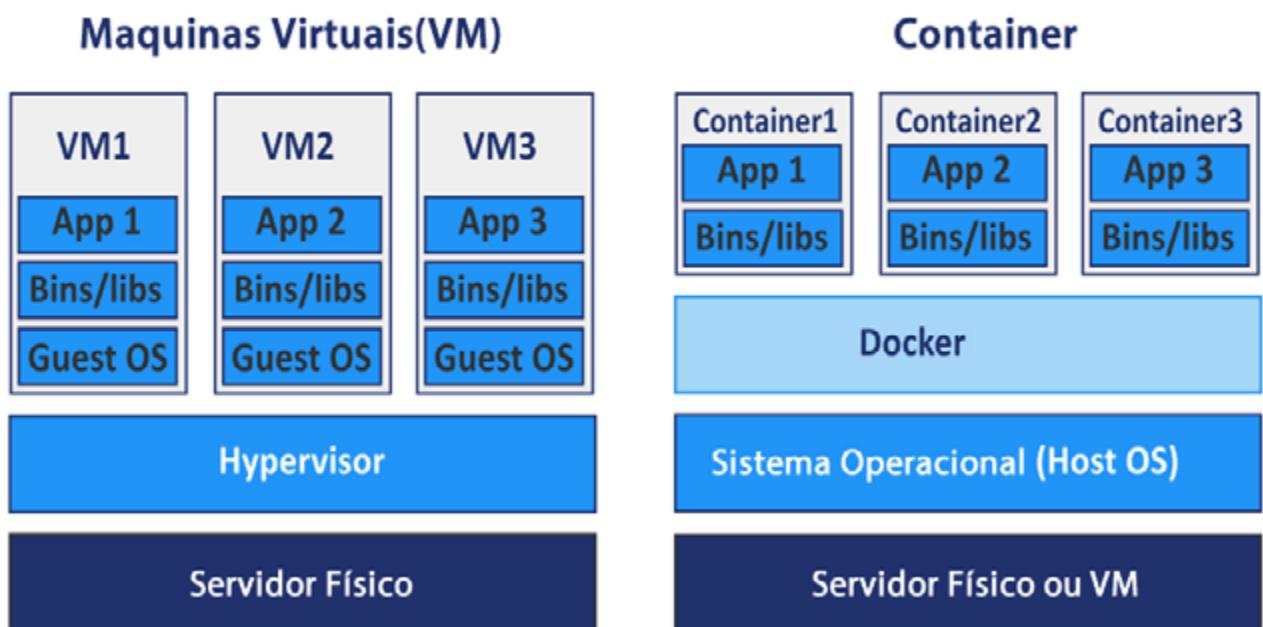
Fonte: bakhtiar\_zein/ iStock.com.

### 1.3.2 Docker

O ambiente em Docker é indicado para os DevOps, por ser uma plataforma aberta para criação, execução e publicação de containers. Um container é recurso de empacotamento da aplicação e suas dependências de forma padronizada.

Nesse entendimento, o Docker se diferencia da tecnologia de virtualização tradicional, que torna possível o empacotamento da aplicação ou todo ambiente em um contêiner, ou seja, esse processo permite as equipes DevOps terem um ambiente portável em qualquer máquina que tenha Docker instalado.

**Figura 7 – Funcionalidades do Docker**



Fonte: adaptado de Nakivo (2021).

Portanto, essa ferramenta auxilia a equipe DevOps a ter um ambiente desenvolvimento alinhado, evitando problemas de incompatibilidade entre as equipes.

No decorrer dos próximos tópicos de estudo, aprofundaremos mais sobre as tecnologias mencionadas acima. Além disso, estudaremos as seguintes temáticas:

- Ferramentas do DevOps no desenvolvimento Web.
- Adoção de práticas DevOps em equipes de desenvolvimento.

## Referências

FERREIRA, M. B. **Métodos ágeis e melhoria de processos**. 1. ed. [s.l.]: Contentus, 2020.

MUNIZ, A. et al. **Jornada DevOps**: unindo cultura ágil, Lean e tecnologia para entrega de software de qualidade. 1. ed. Rio de Janeiro: Brasport, 2019.

NAKIVO. **Nakivo**, 2021. Disponível em: <https://www.nakivo.com/blog/docker-vs-kubernetes/>. Acesso em: 12 jan. 2022.

SOMMERVILLE, I. **Engenharia de Software**. 10. ed., p. 29-35. [s.l.]: Pearson, 2019.

# **Selecionando e conhecendo ferramentas do DevOps no Desenvolvimento Web.**

Autoria: Anderson Pereira de Lima Jerônimo

Leitura crítica: Stella Marys Dornelas Lamounier



## **Objetivos**

- Analisar as ferramentas no ciclo de vida dos projetos.
- Conhecer as principais ferramentas para uso no DevOps.
- Integrar as ferramentas na cultura DevOps.



## 1. Controle de versionamento

O controle de versão é algo importante em projetos atuais, onde as equipes, sejam de desenvolvimentos e/ou de operações, precisam trabalhar, e oferece forma inteligente para organização do projeto, por meio do armazenamento em servidor específico com os históricos das versões do projeto.

Esses históricos, do controle de versão, facilitam o acompanhamento de toda equipe sobre andamento dos projetos e permitem que outros membros da equipe possam desenvolver em paralelo, fazendo testes necessários, além de poder subir para servidor todas as modificações da equipe, para que o servidor possa gerar uma nova versão atualizada.

Com isso, as equipes que venham trabalhar no projeto podem acessar a versão mais atualizada e fazer uma cópia em seu computador local para darem continuidade ao projeto. Existem momentos, durante o desenvolvimento do projeto, em que os membros da equipe estão editando o mesmo arquivo e, para evitar conflitos nos documentos editados, o controle de versionamento oferece recursos para mesclagem do código. (MUNIZ, *et al.*, 2019). Esse recurso auxilia na análise das alterações feitas por outros membros, escolhendo se deseja acatar as modificações ou não. Após essa análise, o código estará pronto para ser enviado ao servidor centralizado que controla as versões, como ilustra a Figura 1.

**Figura 1 – Controle de versionamento**



Fonte: ribkhan/ iStock.com.

Entretanto, por trás da arquitetura de versionamento, são suportadas algumas formas de desenvolvimento:

- Registro do histórico.
- Colaborando concorrentemente.
- Variações no projeto.

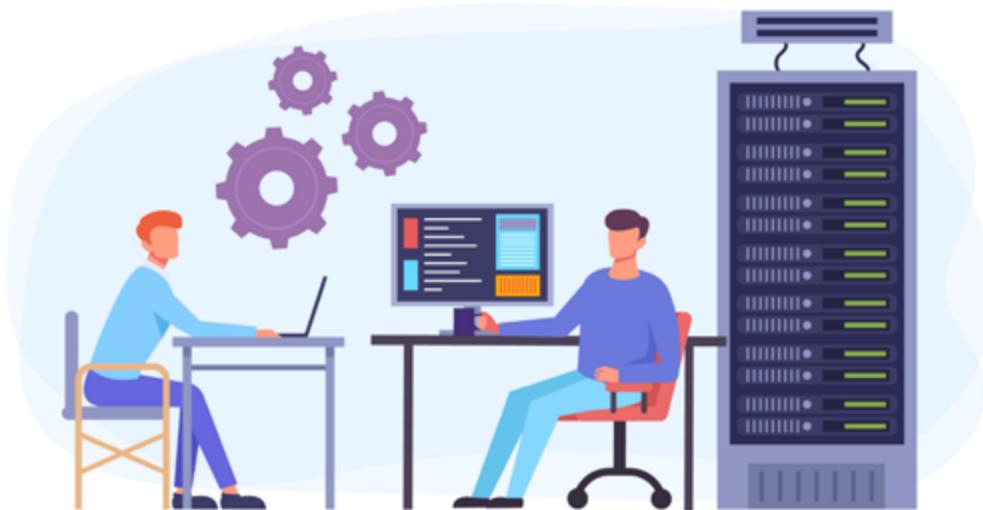
No controle de versão, podemos classificar em dois modelos:

### **Modelo centralizado.**

Constitui em servidor centralizado que atende a maioria dos casos. porém, é mais indicado para equipes menores que estejam na mesma rede local, por ter seu embasamento na arquitetura cliente-servidor, conforme Figura 2.

Nesta arquitetura cliente-servidor, os clientes enviam solicitações para o servidor, que fica responsável em processá-las e respondê-las (PRESSMAN, 2010). Em virtude disso, caso existam equipes maiores que venham trabalhar fora da rede local, esse modelo centralizado não se torna interessante.

**Figura 2 – Modelo centralizado**



Fonte: IrinaGriskova/ iStock.com.

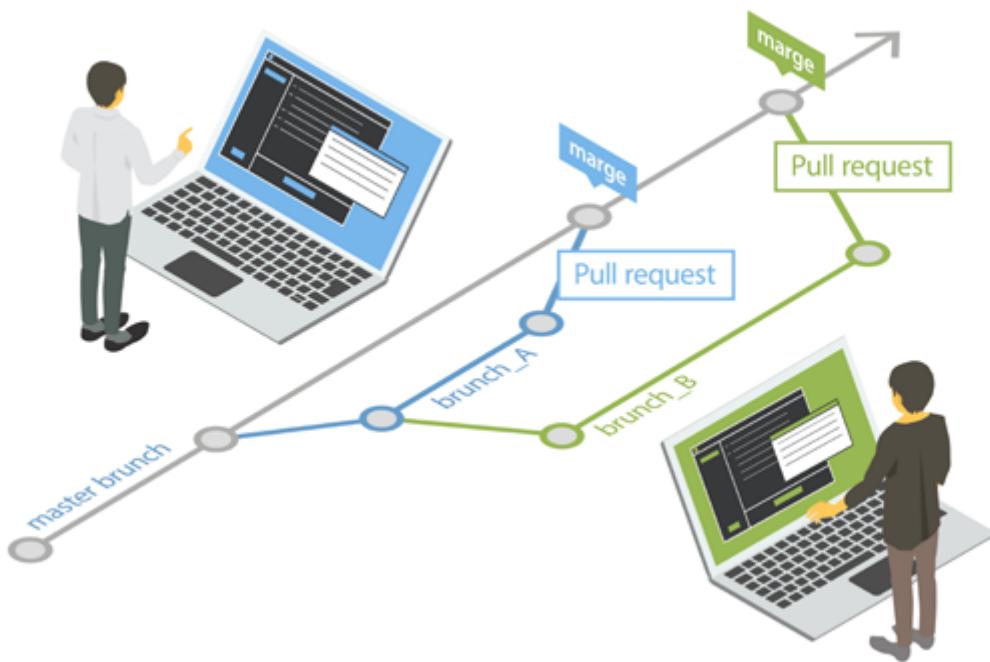
## **Modelo distribuído**

A estrutura de versionamento distribuído é o modelo mais indicado para equipes maiores, que trabalham em lugares diferentes. Nesse modelo, cada membro da equipe possui um servidor próprio acoplado à sua área de trabalho, tendo como principais vantagens: velocidade e robustez ao desenvolvimento não linear.

O modelo GIT trabalha com esse modelo distribuído, que permite aos membros terem seus repositórios próprios, envolvendo todas

as alterações de forma local, não envolvendo uma comunicação dependente com servidor remoto centralizado.

**Figura 3 – Modelo distribuído**



Fonte: interermit/ iStock.com.

## 1.1 GIT

O GIT é um sistema de controle de versionamento distribuído, *open source* e distribuído pela licença GNU GPLv2, criado por Linus Torvalds, o mesmo criador do sistema operacional Linux (MONTEIRO, *et al.*, 2021).

No início, seu propósito era gerenciar o controle de versões do Kernel Linux entre os desenvolvedores, mas o sucesso foi tanto que o uso do sistema foi incentivado para diferentes projetos. Entre suas principais funcionalidades, temos: a velocidade e suporte ao desenvolvimento linear (ramificações, *branches*) e capacidade de suportar projetos grandes, como, por exemplo: kernel do Linux (MONTEIRO, *et al.*, 2021).

O interessante nesse sistema é que tem flexibilidade, onde cada membro da equipe pode contribuir para outros repositórios e, ao mesmo tempo, manter esses repositórios abertos, em que outras pessoas que estejam fora da equipe possam contribuir ou se espelhar no projeto, permitindo um ambiente colaborativo.

O software GIT é preferencialmente usado no desenvolvimento, mas nada impede seu uso para qualquer tipo de arquivo. Isso porque armazenará cada edição feita no arquivo, fazendo histórico do arquivo e registrando as suas respectivas versões.

## 1.2 Principais operações de versionamento

Neste tópico, conheceremos as principais operações de versionamento, presente na maioria dos sistemas de controle de versão, conforme o Quadro 1.

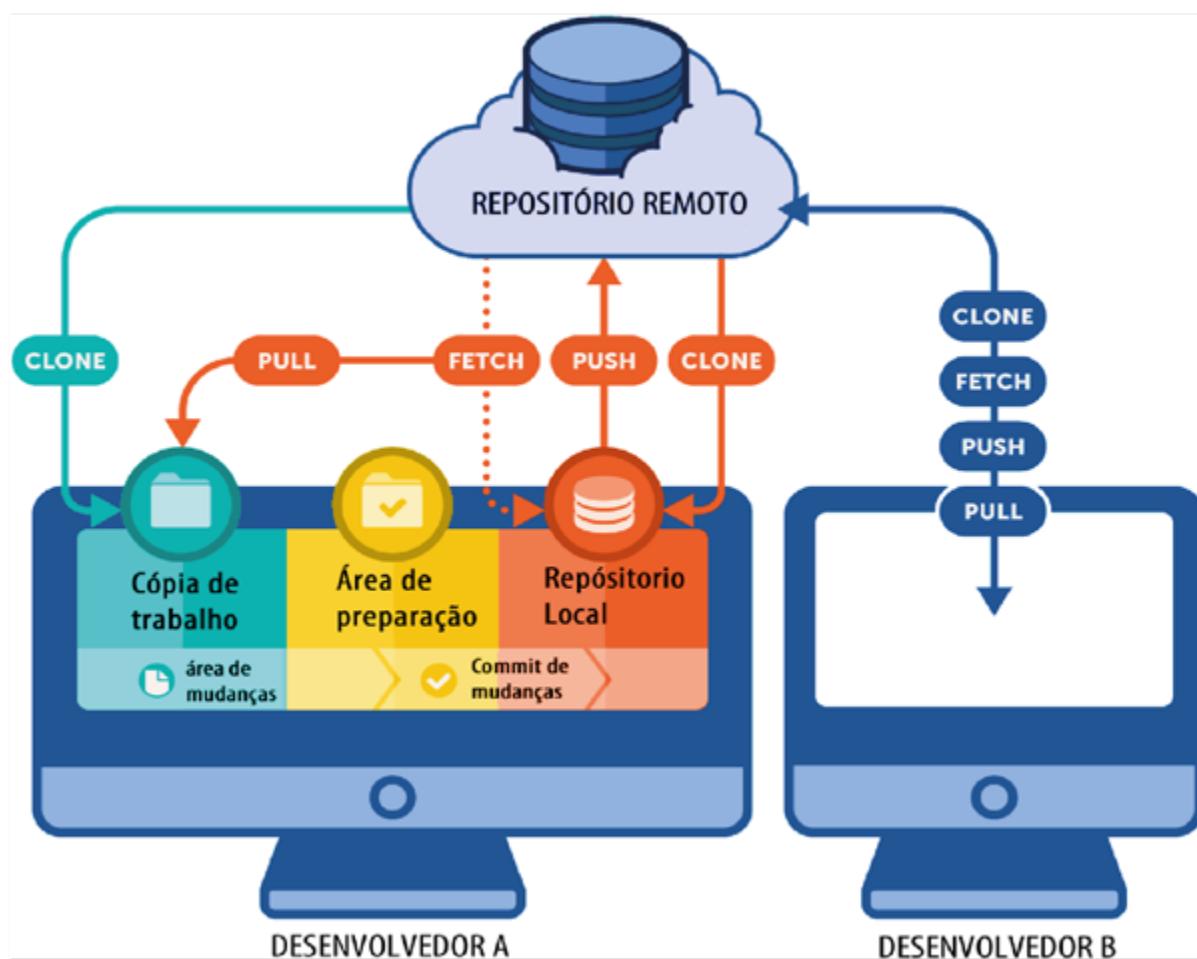
**Quadro 1 – Operações do versionamento**

Operações	Descrição
<i>Check-out.</i>	Essa operação permite recuperar uma versão específica do projeto ou de um arquivo isolado.
<i>Commit.</i>	Criação de uma nova versão do projeto no servidor (repositório).
<i>Revert.</i>	Permite, à equipe, descartar as mudanças realizadas na máquina local, recuperando a mesma versão do servidor (repositório).
<i>Diff.</i>	Faz a comparação do arquivo na máquina local com qualquer outra versão do servidor (repositório).
<i>Delete.</i>	Permite a exclusão de um arquivo do servidor (repositório). Quando as máquinas locais realizarem um update, o arquivo será efetivamente excluído no servidor.

<i>Lock.</i>	Realiza o travamento do arquivo específico, no qual nenhum outro membro da equipe possa modificá-lo
--------------	---

Fonte: elaborado pelo autor.

**Figura 4 – Operações GIT**



Fonte: adaptado de Prajapati (2020).

Podemos perceber que o repositório remoto evita a perda de dados, entretanto, os arquivos que estiverem na máquina local, caso haja algum problema, não poderão ser recuperados (MONTEIRO, *et al.*, 2021). Para cada tipo de controle de versão (centralizado ou distribuído), existem operações básicas.

## Controle de versão centralizado:

- *checkout*.
- *commit*.
- *update*.

## Controle de versão distribuído:

- *clone*.
- *commit*.
- *update*.
- *pull*.
- *push*.

Além dessas operações citadas, o GIT permite a possibilidade de separar as modificações em caminhos diferentes para cada equipe. Esse caminho é conhecido como *branch*, no controle de versionamento. Vale salientar que, embora seja simples o uso de *branches*, se tiver necessidade de criar várias ramificações, será mais difícil de gerenciá-los.

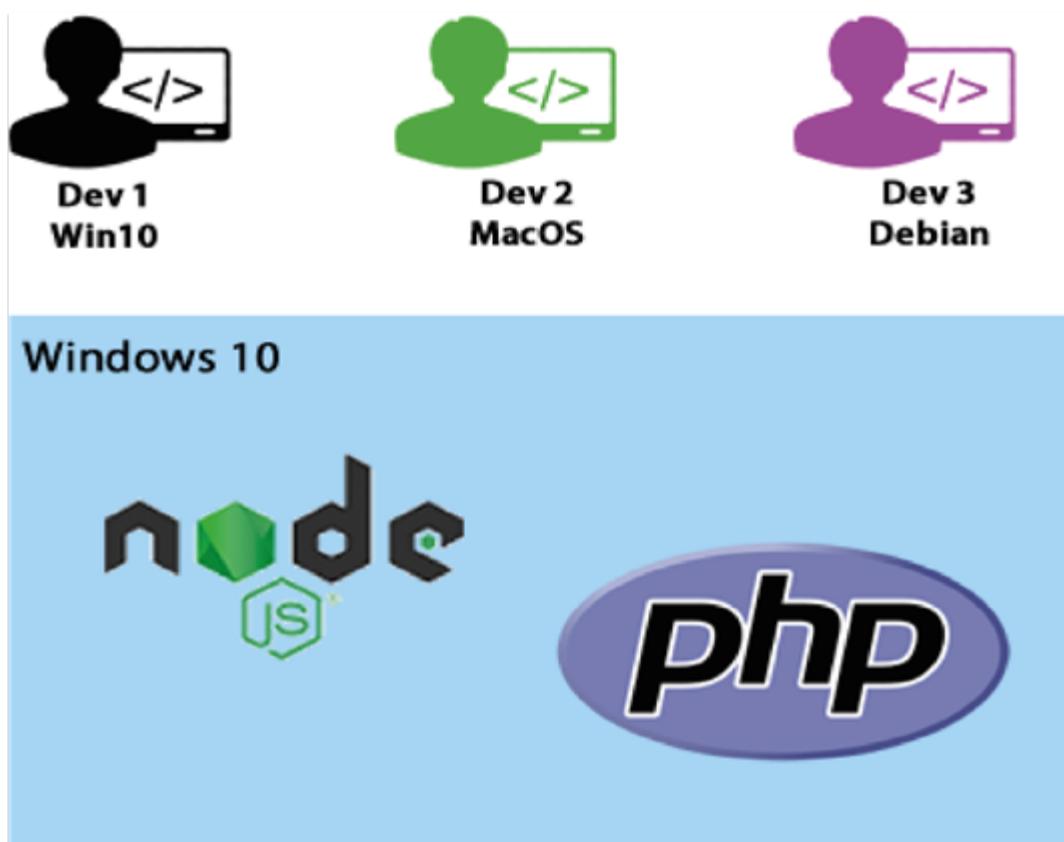
Portanto, o uso desse sistema é essencial para as equipes DevOps, no tocante ao desenvolvimento colaborativo, monitoramento do código e a escalabilidade dos projetos (MUNIZ, et al., 2019, p. 23).

## 1.3 Docker

Analisaremos o seguinte cenário: imagine que você tenha um ambiente Windows com API em Nodejs, com PHP instalado e precisa compartilhar com outros membros da equipe, que trabalham com outros sistemas

operacionais, como: Debian e MacOS. Nessas situações, outras pessoas precisam deixar a máquina com configuração mais próxima possível do ambiente proposto e, às vezes, não é possível devido às versões diferentes das bibliotecas, *frameworks* específicos de um sistema operacional, conforme ilustra a Figura 5.

**Figura 5 – Arquitetura Docker**

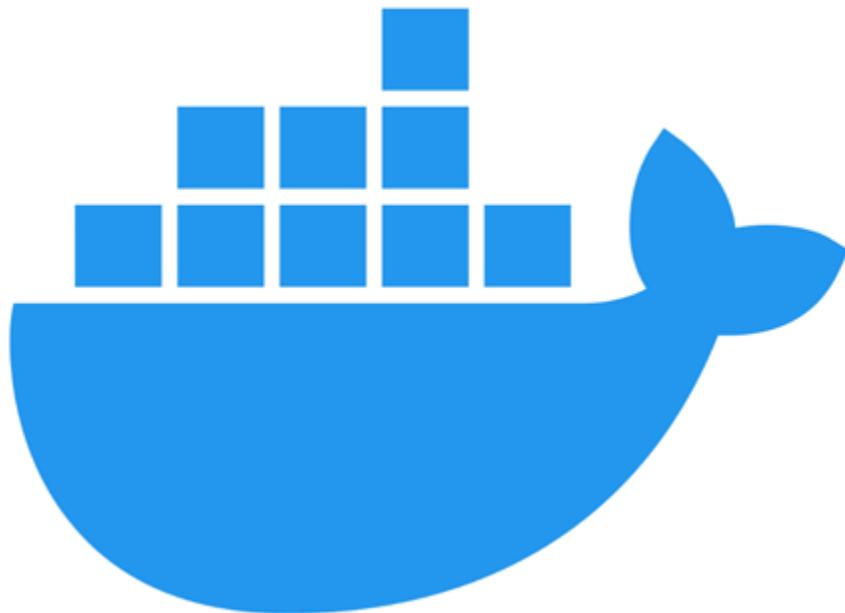


Fonte: elaborada pelo autor.

Devido a este motivo, o Docker surgiu pela necessidade das equipes terem um ambiente de desenvolvimento portável, independentemente do sistema operacional hospedeiro. Essa tecnologia oferece containers virtuais, que empacotam suas dependências para dentro de um contêiner e, a partir desse momento, o container se tornará portável para ser utilizado em qualquer outro lugar que tenha o Docker instalado, ou seja, essas máquinas podem ser locais ou servidores.

O Docker é uma solução que viabiliza a modificação ou substituição de componentes conflitantes de determinado sistema (MONTEIRO, et al., 2021, p. 33). Por meio dele, podemos construir pacotes de sistemas em unidades padronizadas (containers), que já possuem tudo que um sistema precisa para ser acessado.

**Figura 6 – Símbolo do Docker**



Fonte: Oleg Mishutin/ iStock.com.

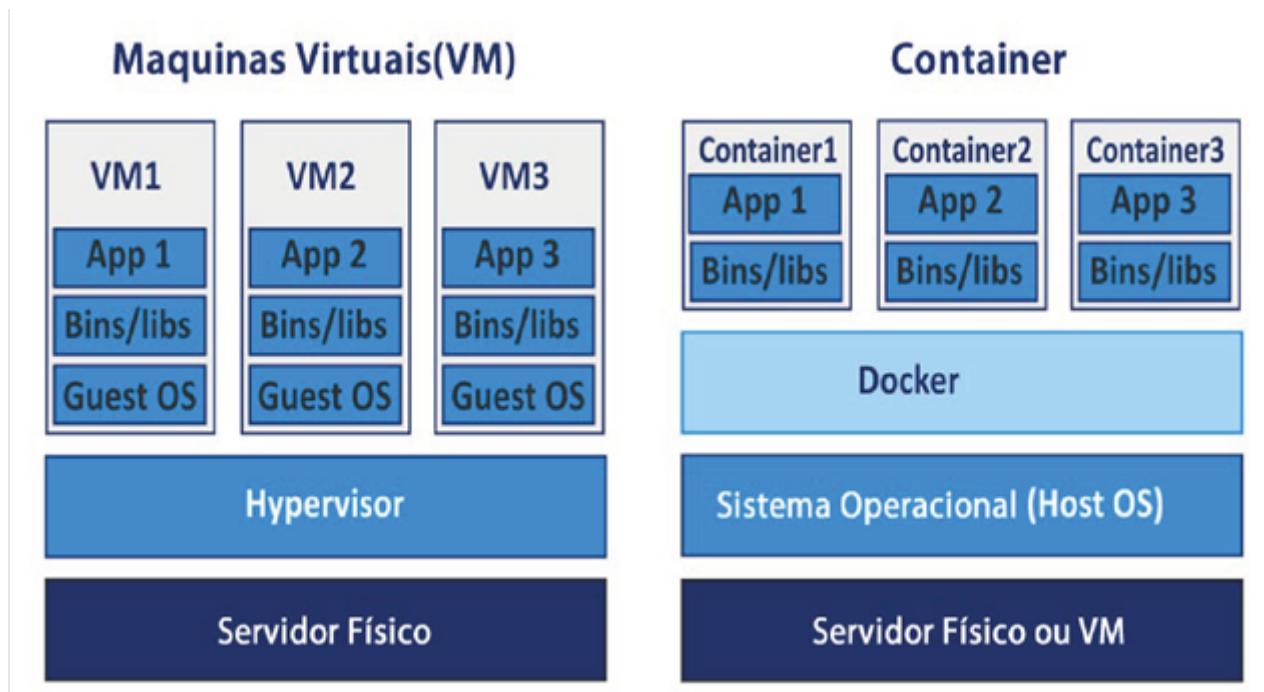
Todas essas configurações dos containers são definidas apenas uma vez no universo Docker e esse procedimento é chamado de criação da imagem. Após essa imagem pronta, você poderá compartilhar entre outros desenvolvedores e instalar em diversos containers esse mesmo ambiente.

## 2. Docker e máquinas virtuais

Quando instalamos uma máquina virtual, temos que gerenciar todo ambiente de desenvolvimento relacionado ao servidor do projeto e que envolve a capacidade do hardware hospedeiro no quesito memória, processamento e armazenamento, ou seja, ficando algumas responsabilidades da equipe para deixar o ambiente estável:

- Configuração do sistema operacional.
- Monitorar possíveis atualizações do sistema operacional.
- *Patches* de segurança.

**Figura 7 – Comparativo VM e Containers**



Fonte: adaptada de Nakivo (2021).

O Docker, como utiliza a estrutura de containers, possui uma inicialização mais rápida, pois a tecnologia se utiliza do kernel da máquina hospedeira para conseguir o compartilhamento de recursos

de hardware. Além de alojar nas suas imagens, alguns megabytes de espaço em disco na máquina hospedeira.

No caso das máquinas virtuais, de acordo com o uso, ficam com tamanho maior, alguns gigabytes, podendo comprometer o armazenamento da máquina hospedeira, conforme Figura 7.

## Referências

MONTEIRO, E. R. *et al.* DevOps. **Grupo A**, 2021. Disponível em: <https://integrada.minhabiblioteca.com.br/#/books/9786556901725/>. Acesso em: 13 jan. 2022.

MUNIZ, A. *et al.* **Jornada DevOps:** unindo cultura ágil, Lean e tecnologia para entrega de software de qualidade. 1. ed. Rio de Janeiro: Brasport, 2019.

NAKIVO. **Nakivo**, 2021. Disponível em: <https://www.nakivo.com/blog/docker-vs-kubernetes/>. Acesso em: 12 jan. 2022.

PRAJAPATI, A. **Medium**, 2020. Disponível em: <https://medium.com/mindorks/what-is-git-commit-push-pull-log-aliases-fetch-config-clone-56bc52a3601c>. Acesso em: 13 jan. 2022.

PRESSMAN, R. S. **Engenharia de Software**. 6. ed. São Paulo: McGraw-Hill, 2010.

# Tecnologia de versionamento GIT e de integração continua

Autoria: Anderson Pereira de Lima Jerônimo

Leitura crítica: Stella Marys Dornelas Lamounier



## Objetivos

- Conhecer os fundamentos do controle de versionamento.
- Analisar os comandos do GIT
- Conhecer os benefícios da integração contínua.

## ► 1. Tecnologia GIT

Nos projetos, quando a equipe não conhece as ferramentas de versionamento, acaba criando cópias do mesmo projeto. Nesses casos começa a criar vários diretórios, ocasionando, na maioria das vezes, o risco de apagar algum arquivo importante do projeto. Caso isso ocorra, fica difícil recuperá-lo. Há também uma outra dificuldade quando não há controle de versão no projeto, no quesito compartilhamento do código. Nesse cenário, a equipe acaba trabalhando com várias cópias do mesmo projeto, dificultando a integração para gerar um único projeto (MONTEIRO; CERQUEIRA; SERPA, ; 2021).

Essas situações, citadas acima, refletem a falta de conhecimento das equipes de desenvolvimento no que tange à falta de uso da tecnologia de versionamento. Diante desse contexto, podemos destacar a ferramenta do GIT, que faz parte da tecnologia de versionamento.

A ferramenta GIT é um sistema de controle de versão de arquivo muito requisitada pelas equipes DevOps, para controlar as modificações do projeto e arquivos de código fonte, facilitando todo processo de gerenciamento de projeto.

**Figura 1 – Tecnologia GIT**



Fonte: itjo/ iStock.com.

Uma característica relevante na ferramenta GIT é quando mencionamos o processo de compartilhamento ao invés de criar várias cópias do projeto para compartilhar. No GIT, concentra-se todo código fonte em único projeto, evitando perdas de código fonte, permitindo melhor gerenciamento e compartilhamento entre os membros da equipe DevOps (MUNIZ; SANTOS; MOUTINHO; IRIGOYEN, 2019).

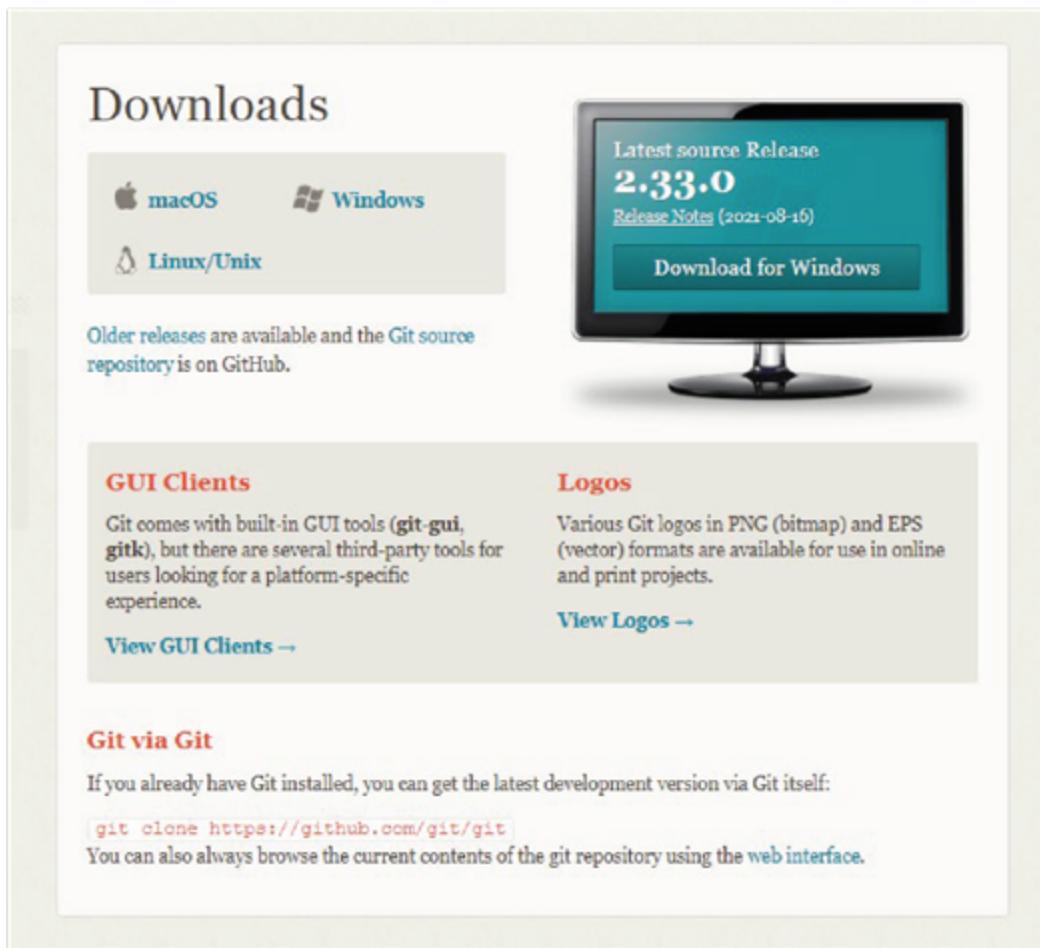
## **1.1 Instalação do GIT**

Entre as ferramentas de versionamento do mercado, o GIT se destaca por ser multiplataforma, ou seja, pode ser instalado em qualquer sistema operacional, seja Windows, Linux e MacOs, tendo um processo de instalação simplificado, mas cada sistema operacional possui uma instalação diferenciada.

### **Instalação no Windows**

A instalação no ambiente Windows é muito simples, apenas precisa baixar a versão mais recente da ferramenta GIT para versão Windows, que podemos obter por meio do site GIT.

## Figura 2 – Print de tela da instalação do GIT



Fonte: print de tela de GIT (2021).

Após o download, executaremos o *setup* e seguiremos as instruções até o fim. Ao chegar no final da instalação, verificaremos se está funcionando corretamente. Como estamos trabalhando no Windows, precisamos abrir o cmd ou PowerShell, e executar o comando para confirmar se houve a instalação corretamente, conforme a Figura 3.

## Figura 3 – Versão instalada

```
git --version
```

Fonte: elaborada pelo autor.

O retorno da sua execução trará a versão instalada no sistema operacional: **git version 2.X.X.**

## Instalação no Linux

No caso sistema operacional Linux, basta abrir o terminal e digitar os comandos conforme ilustra a Figura 4:

**Figura 4 – Atualização dos pacotes e instalação**

```
1 sudo apt update  
2 sudo apt install git
```

Fonte: elaborada pelo autor.

Com isso, o GIT será instalado normalmente para testar se ocorreu tudo certo. Basta abrir o terminal novamente e digitar o comando conforme a Figura 5.

**Figura 5 – Versão instalada**

```
git --version
```

Fonte: elaborada pelo autor.

Como no ambiente Windows, o retorno da execução será exibido na versão instalada no sistema operacional Linux.

## Instalação do macOS:

No sistema operacional do macOS, pode ser feito de várias formas a instalação do GIT, podemos usar Homebrew, Xcode ou próprio instalador. Caso tenha o Xcode já instalado, o GIT já vem embutido automaticamente por meio do Xcode. Geralmente, a versão mantida pela Apple pode ser desatualizada. Para garantir uma versão mais

recente do GIT no macOS, o mais recomendado é a utilização do Homebrew.

O Homebrew é um gerenciador de pacotes para o macOS. Também possui uma instalação muito simplificada, por meio do site do Homebrew. Após instalação do Homebrew, basta abrir o terminal do macOS e digitar o comando conforme a Figura 6.

**Figura 6 – Instalando Homebrew**

```
$ brew install wget
```

Fonte: elaborada pelo autor.

## 1.2 Reppositórios

Na tecnologia de versionamento, temos um local chamado repositório, que permite o armazenamento do código fonte do projeto. Além do armazenamento, é feito o controle de versão. Como o GIT trabalha em modo distribuído, temos dois tipos de repositórios:

### **Local:**

Dispõe de um repositório local, onde são armazenados e versionados sem necessidade de um servidor externo.

### **Remoto:**

Trata-se de um servidor remoto que dispõe de um repositório, onde os códigos fontes são versionados e armazenados. Podemos citar alguns serviços web mais conhecidos que trabalham nessa modalidade, tais como: **GITHUB, GITLAB e BITBUCKET**.

## 1.3 Estados dos arquivos

No controle de versão, utilizando a ferramenta GIT, possuímos basicamente três estados:

### **Modificado:**

Neste estado, acontece quando o arquivo possui alguma alteração em seu conteúdo, seja adicionado ou removido.

### **Preparado:**

O estado preparado é a fase quando adicionamos ao versionamento utilizando o comando, conforme Figura 7.

**Figura 7 – Adicionando na stage**

```
1 git add .  
2
```

Fonte: elaborada pelo autor.

Por meio deste comando, adicionamos arquivos e damos permissões para que o GIT possa fazer o versionamento dos arquivos.

### **Consolidado:**

Nesse estado, após os arquivos serem modificados e preparados, o próximo passo é serem consolidados. Nessa ação é utilizado um comando, como ilustra a figura abaixo:

**Figura 8 – Confirmando as modificações**

```
git commit -m
```

Fonte: elaborada pelo autor.

O comando citado acima tem o objetivo de salvar todas as alterações dos arquivos e o versionamento, montando um histórico das últimas alterações do projeto no repositório local. O **git commit**, é um dos principais recursos do GIT, além de salvar, podemos também desfazer algumas alterações, essa ação chamamos de **rollback**.

Segue quadro abaixo com alguns comandos utilizados no GIT e nas plataformas de repositório Web:

**Quadro 1 – Comandos no GIT**

Comando	Descrição do comando
<code>git status</code>	Verifica o estado do repositório e a área de <i>staging</i> .
<code>git status -s</code> <code>git status -short</code>	Status resumido.
<code>git push origin main</code>	Envia as alterações para o repositório remoto, onde <i>origin</i> é o <i>remote</i> e <i>master</i> o <i>branch</i> .
<code>git pull origin</code>	Baixa <i>commits</i> do repositório remoto.
<code>git fetch origin</code>	Atualiza as referências com um repositório remoto.

Fonte: elaborado pelo autor.

## 1.4 Plataformas de armazenamento de código

No mercado, existem três plataformas de controle de versão, que ficaram popularmente conhecidas entre os desenvolvedores DevOps: o GitHub, GitLab e o BitBuket.

Esses serviços de armazenamento de código na web são ferramentas indispensáveis para as equipes DevOps, por meio desses serviços é possível facilitar o trabalho das equipes em disponibilizar uma entrega contínua de forma mais simplificada, pois cada membro da equipe terá a possibilidade trabalhar em um repositório único, facilitando o

gerenciamento do código, possibilitando sempre trabalhar com código mais recente.

**Figura 9 – Plataformas de armazenamento**



Fonte: elaborada pelo autor.

## 1.5 Comunicação

A ferramentas Web, que utilizam o GIT, trabalham com dois protocolos de comunicação: o HTTPS e o SSH para transferir os arquivos para as plataformas de armazenamento de código.

No caso do protocolo HTTPS, é mais prático, pois não precisa de nenhuma configuração prévia para estabelecer a comunicação, bastando apenas ter o link e a permissão para ter acesso ao repositório.

A comunicação com protocolo SSH é um tipo mais complexa do que a comunicação com protocolo HTTPS. Nela, é necessário que haja a geração de chave criptografia SSH, que seja adicionada no perfil da plataforma (GitHub, GitLab ou Bitbucket).

**Figura 10 – Comunicação GIT**

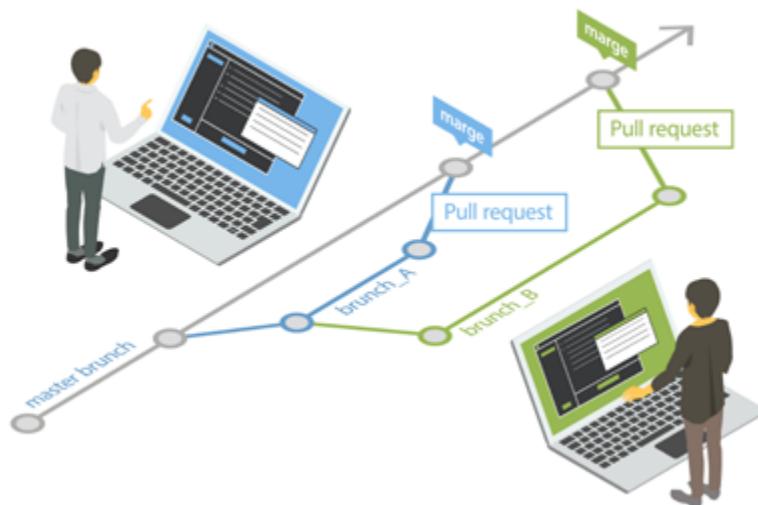


Fonte: MohammedHaneefaNizamudeen/ iStock.com.

## 1.6 Branch

Na tecnologia GIT, temos o *branch*, que serve como um ponteiro onde estão todas as alterações dos arquivos do projeto. É interessante seu uso no quesito de adicionar uma nova funcionalidade ou até mesmo corrigir erros, sem prejudicar diretamente a versão estável do sistema.

**Figura 11 – Ciclo do *branch***



Fonte: interemit/ iStock.com.

Com os *branches*, podemos criar uma linha de *commits* e trabalhar com diferentes linhas de desenvolvimento. Pensamos nessas linhas de desenvolvimento como árvore, onde cada ramificação vai se espalhando a partir de um ponto base, ou seja, essas ramificações são consideradas como *branch*.

Uma curiosidade no *branch*, na situação da referência do GIT por meio de HEAD, ou seja, no GIT essa referência acontece de forma dinâmica, apontando para último *commit* do *branch* atual. Diferente de outros sistemas de versionamento, a criação de vários *branches* não comprometerá o desempenho e nem o tamanho.

Veja, a seguir, os principais comandos do *branch* no GIT:

### **Quadro 2 – Operações do *branch***

<b>Comando</b>	<b>Descrição</b>
<i>git checkout -b nova-branch</i>	Criar uma nova <i>branch</i> .
<i>git checkout master</i>	Alterar entre as <i>branches</i> .
<i>git checkout -</i>	Voltar para a <i>branch</i> anterior.
<i>git branch -d branch</i>	Remover uma <i>branch</i> local.
<i>git push —delete origin branch</i>	Remover a <i>branch</i> remota.

Fonte: elaborada pelo autor.

Podemos dividir os *branches* em alguns tipos:

#### **Branch local:**

É um *branch* criado no repositório local, onde o usuário pode enviá-lo ou não para outro repositório remoto. Com esse *branch*, caso seja enviado para outro repositório, passa a ser público. Uma dica: na maioria das vezes, o *branch* pode embasar outros projetos e, nesses casos, é interessante ter um cuidado ao modificar o histórico do *commit*.

## **Branch temporário:**

Um tipo de *branch* quando possa ter alguma funcionalidade específica, logo na sequência, pode ser combinado com *branch permanente*. No caso do *branch* permanente, está presente em todo ciclo de vida do projeto. Dependendo da estratégia adotada pela equipe DevOps, essa combinação pode ser feita utilizando método *workflow* baseado em *merge* ou *rebase*.

## **1.7 Integração contínua**

O processo de integração contínua tem origem na metodologia ágil XP (*Extreme Programming*). Esse tipo de processo possui como diferencial a possibilidade da equipe desenvolvimento fazer uma integração e/ou melhoria no desenvolvimento do código no projeto principal, sendo feito com maior agilidade. Esse processo permite que aplicações possam se adequar às tendências do mercado com mais rapidez e economicidade de tempo nas integrações.

Podemos citar o software *Jenkins*, que tem papel de orquestrar todo o *pipeline* de entrega de software, fazendo a entrega contínua, juntamente com uma cultura DevOps, que acelera drasticamente a entrega de software. O *Jenkins* é a solução mais amplamente adotada para entrega contínua, por possuir uma extensibilidade e boa aceitação entre comunidade.

A interação da comunidade importante que permite ao *Jenkins* mais de 1.700 *plug-ins* que se integre a virtualmente a qualquer ferramenta, incluindo todas as melhores soluções usadas em todo o processo de entrega contínua. (JENKINS, 2021).



## Referências

- GIT. **GIT**, 2021. Disponível em: <https://git-scm.com/downloads>. Acesso em: 13 jan. 2022.
- JENKINS. Jenkins, 2021. Disponível em: <https://www.jenkins.io/>. Acesso em: 13 jan. 2022.
- MONTEIRO, E. R.; CERQUEIRA, M. B.; SERPA, M. S. *et al.* **DevOps**, 2021. Disponível em: [https://integrada\[minhabiblioteca\].com.br/#/books/9786556901725/](https://integrada[minhabiblioteca].com.br/#/books/9786556901725/). Acesso em: 13 jan. 2022.
- MUNIZ, A.; SANTOS, R.; MOUTINHO, R. *et al.* **Jornada DevOps: unindo cultura ágil, Lean e tecnologia para entrega de software de qualidade**. 1. ed.). Rio de Janeiro: Brasport, 2019.
- PRAJAPATI, A. **Medium**, 2020. Disponível em: <https://medium.com/mindorks/what-is-git-commit-push-pull-log-aliases-fetch-config-clone-56bc52a3601c>. Acesso em: 13 jan. 2022.
- PRESSMAN, R. S. **Engenharia de software**. 6. ed.). São Paulo: McGraw-Hill, 2010.

# Adoção das Práticas DevOps em Equipes de Desenvolvimento.

Autoria: Anderson Pereira de Lima Jerônimo

Leitura crítica: Stella Marys Dornelas Lamounier



## Objetivos

- Analisar algumas práticas importantes no desenvolvimento.
- Conhecer fundamentos do Docker.
- Aprender gerenciar imagens para ambiente de desenvolvimento.



## 1. Práticas no desenvolvimento

A metodologia DevOps é importante nas organizações que o adotam, justamente na necessidade de unir equipes supostamente opostas que são os operadores e os desenvolvedores (MUNIZ; SANTOS, *et al.*, 2019).

**Figura 1 – Trabalho em equipe**



Fonte: PeopleImages/ iStock.com.

Cada equipe possui seu papel dentro da organização e, geralmente, a equipe de desenvolvimento fica responsável em desenvolver de forma ágil às demandas impostas para que haja uma entrega rápida, com foco nas mudanças que o sistema e/ou aplicativo venha surgir. Enquanto isso, a equipe de operadores desempenha um papel fundamental, trata em manter o sistema sempre disponível, tornando as aplicações cada vez mais escaláveis, com a possibilidade de ter uma frequência maior de entregas contínuas.

As práticas DevOps vem com a finalidade de unir esforços das equipes de desenvolvimento, de operações e quando há necessidade também da equipe de segurança. Quando as três equipes trabalham em conjunto, é usado o termo muito conhecido, chamado de DevSecOps. Com essa orquestração de tarefas e compartilhamento de tarefas, permite acelerar o *deploy* de aplicações de forma rápida e segura, garantindo releases mais estáveis.

Podemos destacar a prática DevOps, pela capacidade de concentrar esforços em padronizar o código fonte desde início até sua fase de implantação, evitando um retrabalho na codificação.

Outro ponto interessante, é que as equipes podem gerenciar todo processo de automação nas configurações, com interesse em fazer análises de vários domínios da aplicação, pois, caso venha surgir algum problema, esse seja resolvido de forma rápida e mais assertiva em sua resolução (MUNIZ, SANTOS, et al., 2019).

Não podemos deixar de citar, nas práticas de desenvolvimento DevOps, o uso das tecnologias de controle de versionamento e suas principais estratégias de ramificação, a elaboração de um canal eficiente para entregas e integrações contínuas, onde é usado um termo inglês chamado de *pipeline*, e disponibilizar microsserviços por meio dos containers padronizados e monitorar os sistemas em tempo de execução, em ambientes de homologação e produção.

## 1.1 Ambiente de containers versus máquinas virtuais

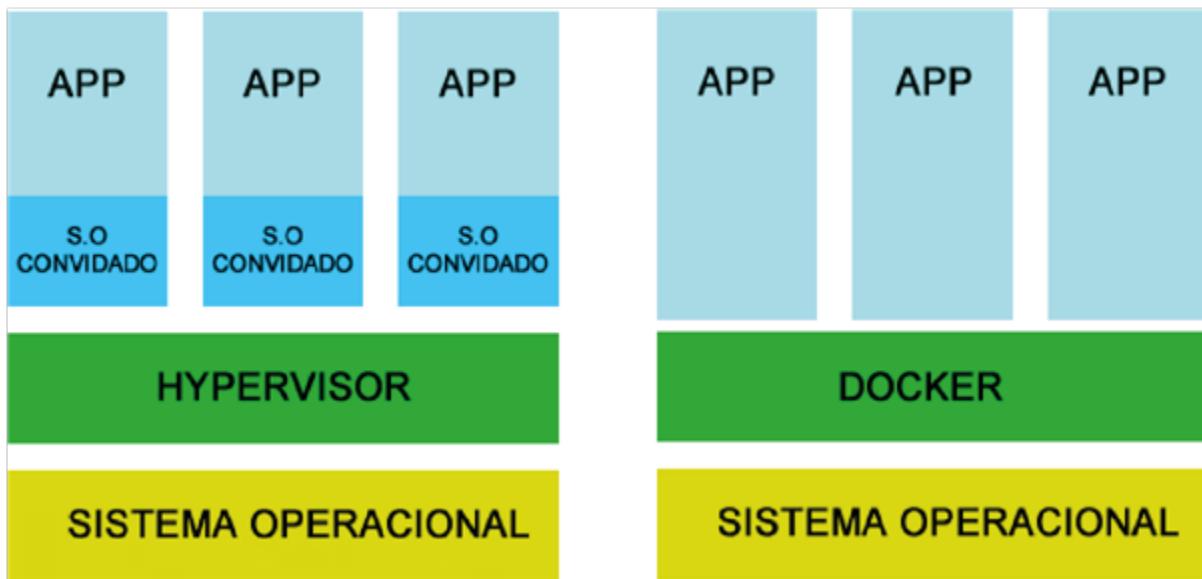
O papel da metodologia DevOps é garantir a produtividade nos processos com uso de métodos ágeis, sem perder a qualidade das operações. Quando a equipe de desenvolvimento possui esse olhar de ter sistemas padronizados desde sua concepção, isso ajuda as equipes de operadores a manter tais funcionalidades de forma mais coesa.

Uma das dificuldades das equipes de desenvolvimento que não trabalham com ambiente de containers, é a dificuldade de compartilhar o ambiente de desenvolvimento com suas devidas dependências para diferentes sistemas operacionais entre seus demais membros, assim como o controle de versionamento que auxiliam que todos possam trabalhar com mesmo código fonte, que concentre todas as alterações em único repositório remoto, no uso containers, que, além de facilitar no ambiente de desenvolvimento, acelera a infraestrutura na fase de produção com *deploy* das aplicações, isto é, fazendo que as aplicações rodem em diferentes plataformas de forma transparente para usuários (MUNIZ, SANTOS, *et al.*, 2019).

Os containers trabalham de formas diferentes das tradicionais máquinas virtuais, ou seja, nas máquinas virtuais, seu desempenho para ser favorável em ambiente de produção dependeár exclusivamente na máquina hospedeira, pois, caso o hardware onde fica a camada de virtualização, conhecida como *hypervisor*, possua pouca memória RAM e/ou processador com poucos núcleos de processamento, comprometerá a usabilidade mais fluida da máquina virtual, tornando-a lenta.

O container possui uma certa vantagem em relação às máquinas virtuais, por não possuir um sistema operacional completo para sua execução. Por utilizar e compartilhar o mesmo *kernel* do sistema operacional hospedeiro, onde estão sendo executados. Conforme vão sendo executados, os containers vão trabalhando de forma semelhante aos processos dos sistemas de operacionais. Esse modo de operação gera uma maior velocidade e, ao mesmo tempo, um modo mais leve do que as máquinas virtuais, podendo executar vários containers em único servidor ao invés de ter várias máquinas de virtuais em mesmo servidor hospedeiro.

**Figura 2 – Máquinas virtuais e o Docker**



Fonte: elaborada pelo autor.

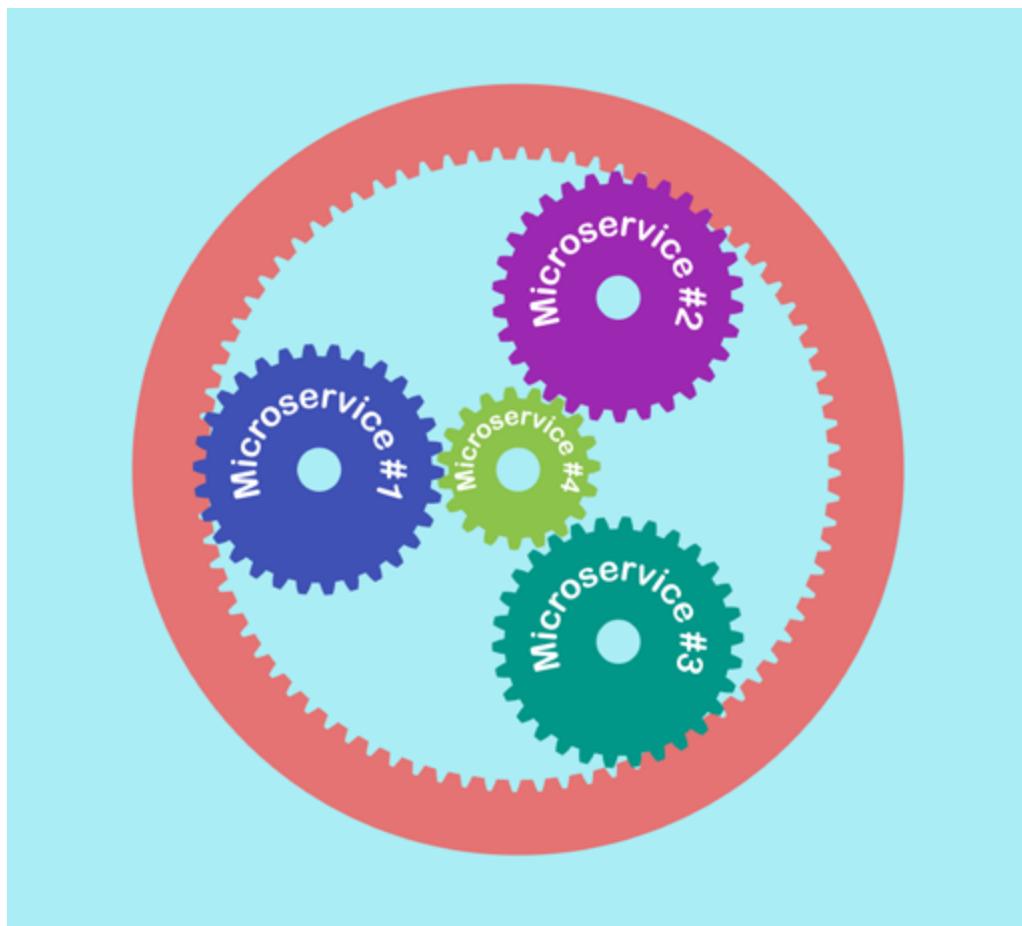
## 1.2 Tecnologia Docker

O Docker é uma plataforma de tecnologia *open source* com boa aceitação no mercado de trabalho, que auxilia no uso de contêineres, assim como temos vários programas que ajudam nas máquinas virtuais, tais como: Oracle VirtualBox e o VmWare. O Docker facilita na construção e no gerenciamento das imagens dos containers, trazendo uma otimização mais eficiente dos recursos a serem trabalhados, por possuírem uma inicialização mais rápida, comparado às máquinas virtuais.

Essa tecnologia permite trabalhar de forma distribuída, utilizando os conceitos de microsserviços. Nesse modo, conseguimos criar um ambiente mais isolado com as dependências específicas para seu funcionamento, podendo criar microsserviços de multilinguagens, por exemplo: um micro serviço para Javascript, onde esse podemos ter uma imagem que der suporte à npm, nodejs, ngnix, e até mesmo um microsserviço Java, que tenha uma imagem dando suporte ao Maven, Tomcat e outros. Portanto, isso garante que cada microsserviço venha

trabalhar de forma isolada e funcione em qualquer ambiente e em qualquer servidor.

**Figura 3 – Microsserviços em conjunto**



Fonte: Oleg Mishutin/ iStock.com.

Podemos também citar uma característica do Docker, que é no fato de facilitar o compartilhamento dos contêineres entre os membros da equipe de desenvolvimento DevOps, bastando apenas a execução de um script com todo descritivo de quais imagens e dependências que precisarão para montar um ambiente de desenvolvimento. Nesse modo apresentado acima, favorece que o sistema seja escalável ao longo do tempo, facilitando a expansão da aplicação.

## 1.3 Instalação do Docker

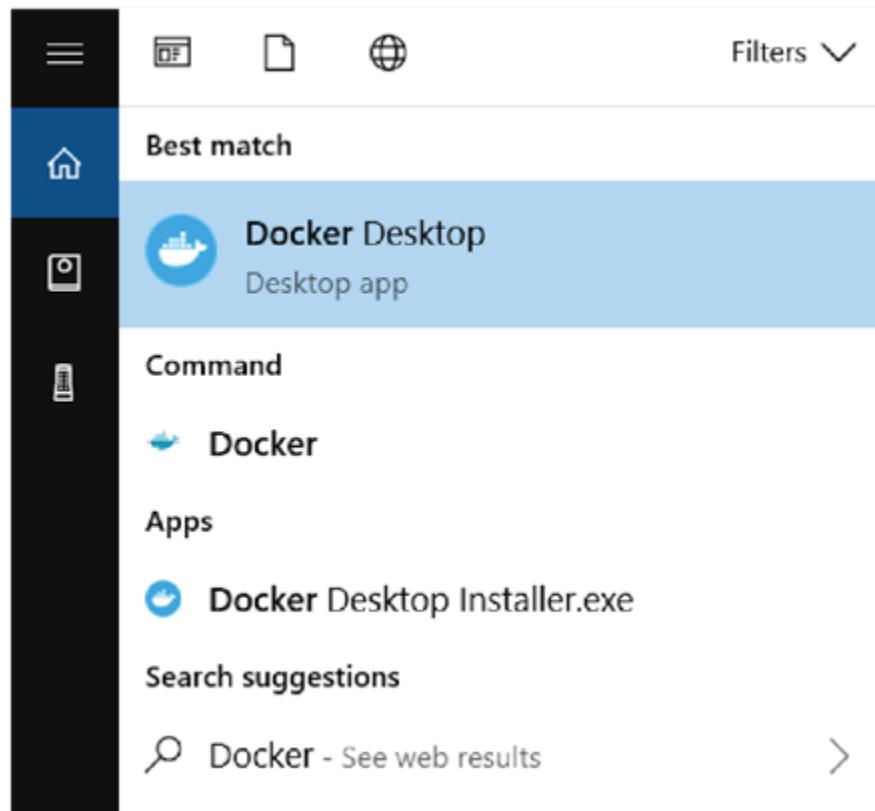
O Docker é composto por dois produtos, a versão *Community Edition* (CE) e a versão *Enterprise Edition* (EE). Na versão CE, a plataforma é gratuita com algumas limitações, que dispõem com recursos mais recentes para desenvolvedores e a imagens no Docker HUB. Já na versão EE, permite uma assinatura da plataforma, que garante recursos adicionais de segurança e orquestração integrada, suporte e tecnologia certificada para infraestrutura e plug-ins. Na plataforma Docker, pode ser instalado em diferentes plataformas de sistemas operacionais, tais como: Linux, Windows e MacOS.

A principal recomendação que a máquina hospedeira possua no mínimo um processador Dual Core, e pelo menos 8GB de memória RAM. Além disso, é interessante possuir um tamanho de armazenamento razoável para baixar as imagens, só a instalação do Docker ocupa quase 2 GB em disco. Após sua instalação, teremos acesso aos comandos do Docker CLI para fazer a instalação e gerenciamento das imagens e dos volumes de persistência.

### Windows:

Para instalação no sistema operacional Windows, é mais recomendado que seja na versão do Windows 10 ou superior. Além disso, precisamos criar uma conta no site do Docker Hub, e essas credenciais serão usadas para ativação e no gerenciamento das imagens. Após o cadastro, você poderá baixar o programa desktop chamado Docker for Windows, na tela Get Docker.

**Figura 4 – Docker no S.O. Windows**



Fonte: print de tela de Docker no Windows, adaptado pelo autor.

Ao instalar o Docker, por meio do programa desktop, em paralelo, é feita a instalação do CLI. Para verificar se ocorreu a instalação corretamente, basta abrir o prompt comando ou Power Shell e, em seguida, ao digitar o comando, conforme Figura 5, podemos ver a versão do Docker instalada na máquina.

**Figura 5 – Versão do Docker**

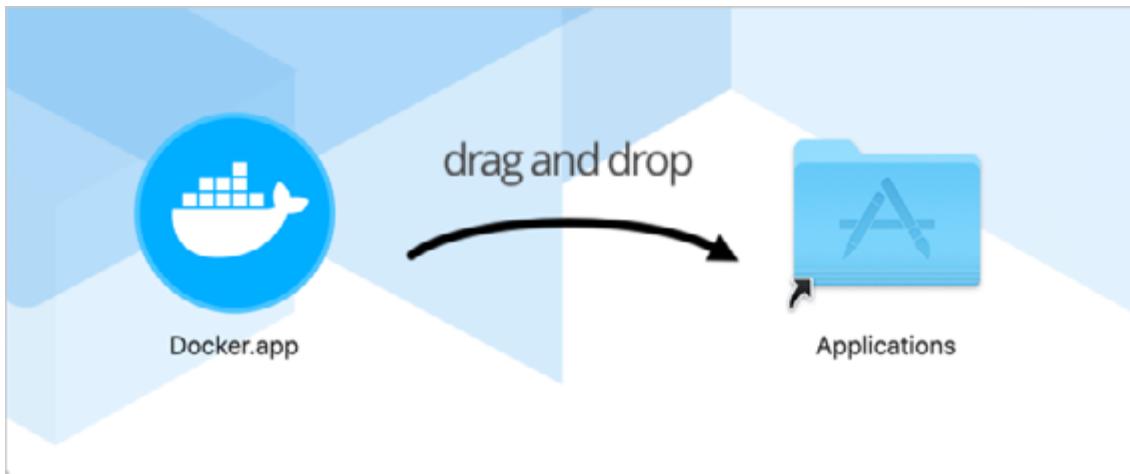
```
docker --version
```

Fonte: elaborada pelo autor.

## MacOS

Na instalação do sistema operacional MacOs, é aconselhável versão do sistema operacional mais recente instalada, e baixar a versão atual do Docker desktop para MacOs.

**Figura 6 – Docker no S.O. MacOs**



Fonte: elaborada pelo autor.

Após instalação, você poderá testar, via terminal, se o Docker CLI está funcionando, por meio do comando semelhante usado no ambiente windows: *docker -version*.

## Linux

No sistema operacional Linux, o Docker oferece suporte para os pacotes *.deb* e *.rpm* das principais distribuições e arquiteturas dos sistemas operacionais Linux, como podemos observar na Figura 7.

**Figura 7 – Pacotes Docker no Linux**

Plataforma	x86_64 / amd64	arm64 / aarch64	braço (32 bits)	s390x
CentOS	✓	✓		
Debian	✓	✓	✓	
Fedora	✓	✓		
Raspbian			✓	
RHEL				✓
SLES				✓
Ubuntu	✓	✓	✓	✓
Binários	✓	✓	✓	

Fonte: print de tela de Docker, no Linux, adaptado pelo autor.

Para demonstrar a instalação no ambiente Linux, escolheremos a distribuição do CentOS, para instalação do Docker. Antes de começar a instalar o Docker, é necessário atualizar os pacotes do Linux, conforme Figura 8.

**Figura 8 – atualização dos pacotes Linux**

```
$ sudo yum update -y
```

Fonte: elaborada pelo autor.

Após a atualização dos pacotes, o próximo passo é adicionar o repositório do Docker para *yum*, e tais comandos podemos observar na figura 9.

## Figura 9 – Adicionando o repositório Docker no Linux

```
$ sudo yum install -y yum-utils  
$ sudo yum-config-manager --add-repo  
https://download.docker.com/linux/centos/docker-ce.repo
```

Fonte: elaborada pelo autor.

Na sequência, removeremos alguns pacotes que venham conflitar com o Docker, conforme figura abaixo.

## Fonte 10 – Remover pacotes

```
$ yum erase podman buildah -y
```

Fonte: elaborada pelo autor.

Após a preparação feita com comandos listados acima, instalaremos de fato o Docker na máquina Linux, por meio do comando abaixo.

## Figura 11 – Instalação do Docker

```
$ sudo yum install docker-ce docker-ce-cli containerd.io -y
```

Fonte: elaborada pelo autor.

Agora, o próximo passo será habilitar o serviço do Docker para que esteja sempre disponível. Sendo primeiro comando para habilitar o serviço, e outro para iniciar, conforme Figura 12.

## Figura 12 – Habilitar e iniciar o serviço Docker no Linux

```
$ sudo systemctl enable docker  
$ sudo systemctl start docker
```

Fonte: elaborada pelo autor.

Por fim, para verificar se ocorreu tudo certo com a instalação e o status do Docker, podemos executar os seguintes comandos, conforme Figura 13.

**Figura 13 – Informações sobre Docker**

```
$ docker info  
$ systemctl status docker
```

Fonte: elaborada pelo autor.

## 1.4 Imagens Docker

Na infraestrutura de Images Docker, podemos definir as imagens como sistemas de arquivos de camadas que ficam uma sobre as outras. Sendo a base para criação de uma aplicação, pode ser desde a base do Ubuntu, como também um Ubuntu com Nginx, Nodejs e Mongo.

Há um serviço chamado de Docker Hub, fornecido pela empresa Docker, para localizar e compartilhar imagens de contêiner com as equipes DevOps. É considerando maior repositório do mundo de imagens de contêineres com uma variedade de fontes de conteúdo. Os usuários obtêm acesso a repositórios públicos gratuitos para armazenamento e compartilhamento de imagens ou podem optar por um plano de assinatura para repositórios privados.

Para criar uma imagem que contenha, por exemplo, um servidor web na máquina onde Docker está instalado, é muito simples, basta digitar o seguinte comando: **sudo docker run —name docker-nginx -p 80:80 nginx**.

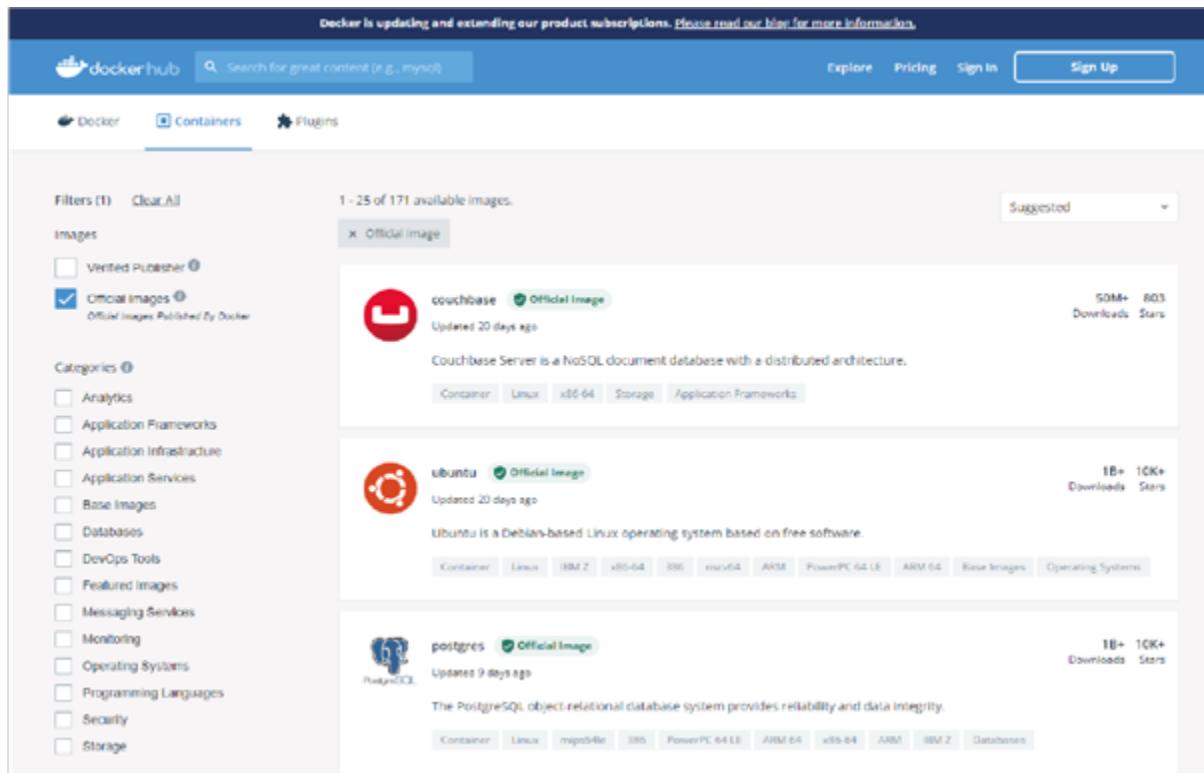
## Quadro 1 – Criação de containers

Comandos	Descrição
run	O comando para criar um container.
—name	Sinalizador é como especificamos o nome do conteiner.
-p	Especifica a porta que estamos expondo no formato -p local-machine-.
port	Internal-container-port. Neste caso, estamos mapeando a porta 80 no contêiner para a porta 80 no servidor.
nginx	Nome da imagem no dockerhub.

Fonte: elaborado pelo autor.

Em questão de segundos, terá um servidor web disponível na porta 80, para hospedar suas aplicações, pois essa imagem chamada *nginx* está disponível no *Dockerhub*, com também outras configurações específicas.

## Figura 14 – Repositório do DockerHub



Fonte: print de tela de DockerHub, adaptado pelo autor.



## Referências

MONTEIRO, E. R. et al. DevOps. **Grupo A**, 2021. Disponível em: <https://integrada.minhabiblioteca.com.br/#/books/9786556901725/>. Acesso em: 13 jan. 2022.

MUNIZ, A. et al. **Jornada DevOps**: unindo cultura ágil, Lean e tecnologia para entrega de software de qualidade. 1. ed. Rio de Janeiro: Brasport, 2019.



---

**BONS ESTUDOS!**