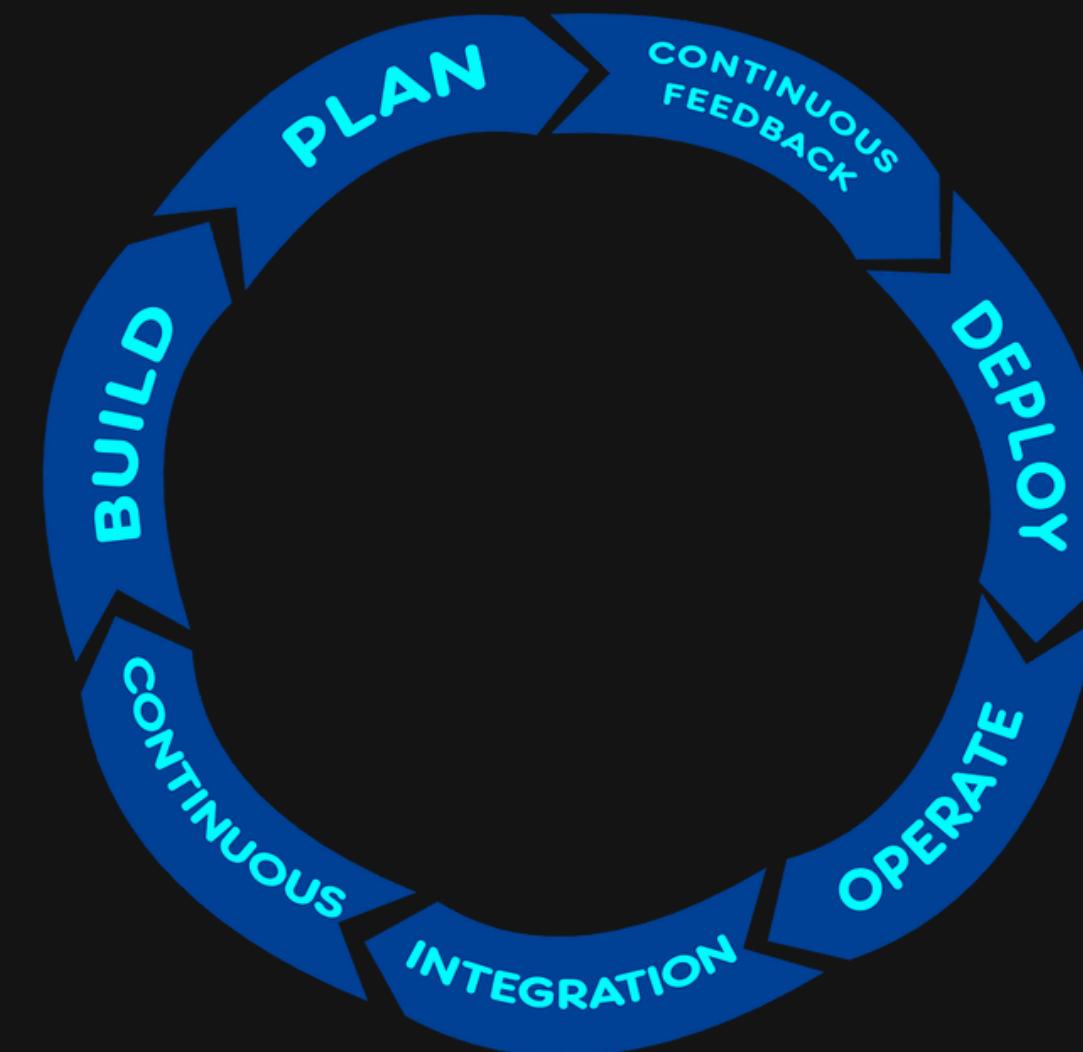


Guia de Informações

# DevOps

A base da automação e colaboração



# ELABORADORES

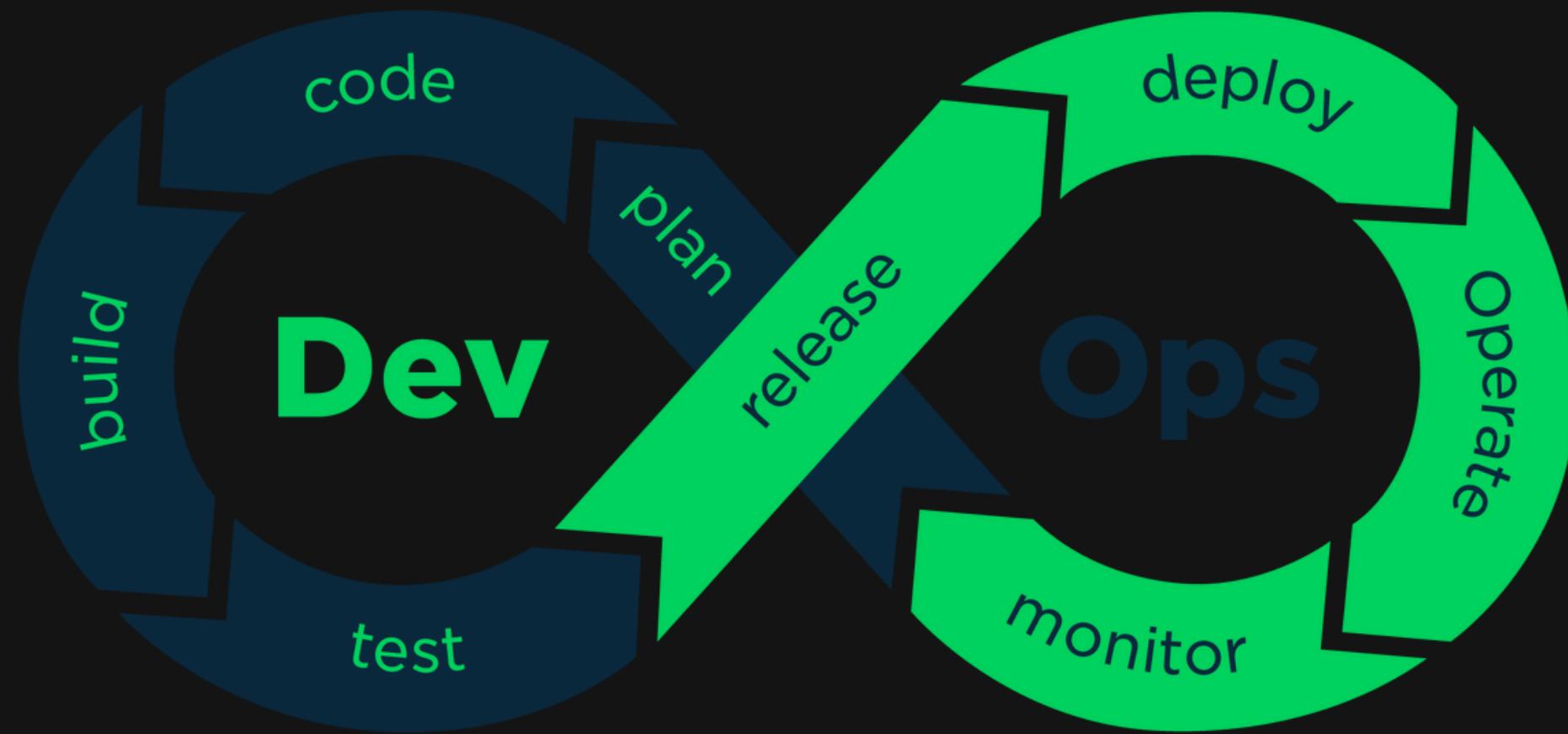


**NELSON FELIPE ANDRADE ARAÚJO**



**JOSÉ LEVY RODRIGUES DA SILVA**

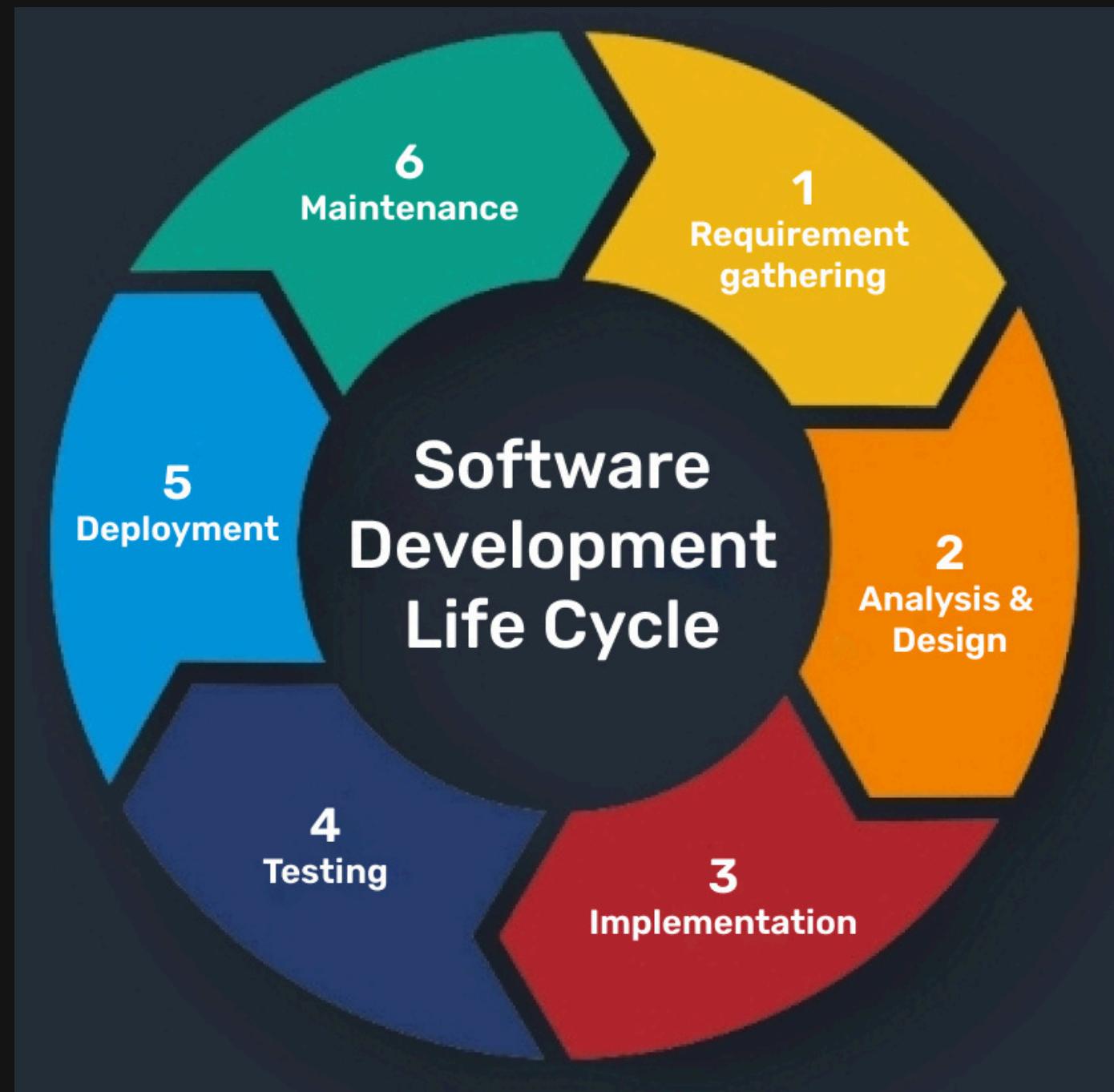
# CONCEITOS DE DEVOPS



# CÍCLO DE VIDA DE UM SOFTWARE

---

- 1. COLETA DE REQUISITOS (REQUIREMENT GATHERING)**
- 2. ANÁLISE E PROJETO (ANALYSIS & DESIGN)**
- 3. IMPLEMENTAÇÃO (IMPLEMENTATION)**
- 4. TESTES (TESTING)**
- 5. IMPLANTAÇÃO (DEPLOYMENT)**
- 6. MANUTENÇÃO (MAINTENANCE)**



# COLETA DE REQUISITOS

---

Levantar requisitos é a fase inicial e a base de todo o projeto. O objetivo é entender o que o software precisa fazer. Envolve coletar todas as **informações** e exigências dos **stakeholders**. O resultado desta fase é, geralmente, um documento que especifica detalhadamente tudo o que o sistema deve ser capaz de realizar.

stakeholders = indivíduos, grupos ou organizações que têm interesse direto ou indireto numa empresa ou projeto.

# ANÁLISE E PROJETO

---

- Análise: A equipe estuda a **viabilidade** dos requisitos que foram levantados e com base neles se organiza.
- Projeto (Design): Cria-se a "planta" do software. Os arquitetos e desenvolvedores definem a **estrutura geral** do sistema, o design da interface, a arquitetura do banco de dados e outras especificações técnicas. O objetivo é ter um plano claro de como o software será construído.

# IMPLEMENTAÇÃO

---

Fase de **codificação**. Os desenvolvedores pegam as especificações de design da fase anterior e começam a escrever o código-fonte do software. É nesta etapa que o software começa a ser construído de fato, transformando o projeto em um **produto funcional**.

# TESTES

---

Após a implementação, o software passa por **testes rigorosos** para garantir sua **qualidade**. A equipe de QA (Quality Assurance) procura por defeitos, bugs e verifica se o software atende a todos os requisitos definidos na primeira fase. Se erros forem encontrados, eles são reportados aos desenvolvedores para **correção**.

# IMPLEMENTAÇÃO

---

Uma vez que o software foi testado e aprovado, ele está pronto para ser lançado. Nesta fase, o software é instalado no ambiente de **produção**, o que significa que ele é **disponibilizado** para os usuários finais. Esta etapa pode envolver a configuração de servidores, bancos de dados e a migração de dados, se necessário.

# MANUTENÇÃO

---

A fase de manutenção começa assim que o software está em produção:

- Correção de bugs: Resolver problemas que não foram encontrados durante a fase de testes.
- Atualizações: Fazer melhorias no sistema ou adaptá-lo a novas necessidades.
- Suporte: Estar disposto a ajudar os usuários a utilizar o software.

O ciclo não termina após o lançamento!

# MODELOS DE CRIAÇÃO DE SOFTWARE

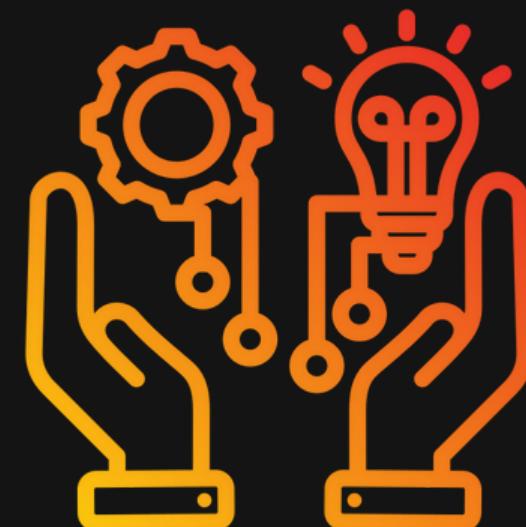
---

## ANTERIORMENTE

- MODELO EM CASCATA (WATERFALL)

## ATUALMENTE

- SCRUM
- KANBAN
- XP



# O QUE É DEVOPS?

---

É uma **cultura** e um **conjunto de práticas** que visam unificar o desenvolvimento de software (Dev) e as operações de tecnologia da informação(Ops). O objetivo principal é **encurtar o ciclo de vida** de desenvolvimento e fornecer uma entrega contínua de software com alta qualidade. Em essência, o DevOps promove colaboração mais direta entre as equipes de desenvolvimento e operações, que tradicionalmente trabalham de forma dividida.



# SURGIMENTO DO DEVOPS

---

O DevOps tem seu surgimento entre os anos de 2007 e 2009, como uma **evolução natural** e uma resposta direta às limitações dos modelos tradicionais de desenvolvimento de software, que não conseguiam atender à crescente necessidade de velocidade e eficiência do mercado. Dessa forma, as equipes de Desenvolvimento (Dev) e Operações (Ops) possuem uma maior interação.

DevOps = Desenvolvimento (Dev) + Operações (Ops)



# FERRAMENTAS DE DEVOPS

---

1. PLANEAMENTO E COLABORAÇÃO: JIRA, CONFLUENCE, TRELLO.
2. CONTROLE DE VERSÃO: GIT, GITHUB, GITLAB, BITBUCKET.
3. INTEGRAÇÃO E ENTREGA CONTÍNUA (CI/CD): JENKINS, GITHUB ACTIONS, CIRCLECI.
4. CONTEINERIZAÇÃO E ORQUESTRAÇÃO: DOCKER, KUBERNETES.
5. INFRAESTRUTURA COMO CÓDIGO: TERRAFORM, ANSIBLE, PUPPET, CHEF.
6. MONITORIZAÇÃO E OBSERVABILIDADE: PROMETHEUS, GRAFANA, ELK STACK , DATADOG, NEW RELIC.
7. COMPUTAÇÃO NA NUVEM (CLOUD COMPUTING): PROVEDORES COMO AWS, MICROSOFT AZURE E GOOGLE CLOUD  
PLATFORM



# CONTEÚDOS DO MINICURSO

---

(Versionamento de código)

**Git & GitHub**



(CI/CD)

**GitHub Actions**



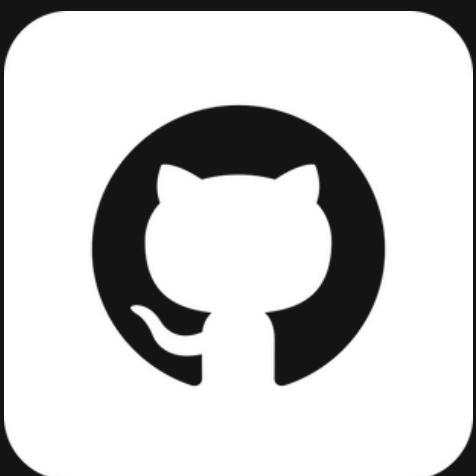
(Conteinerização)

**Docker**

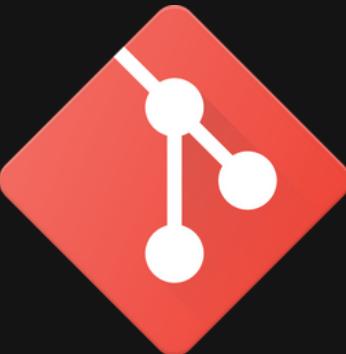




# GIT E GITHUB



# INTRODUZINDO GIT



Git é um Sistema open-source de gerenciamento de versões

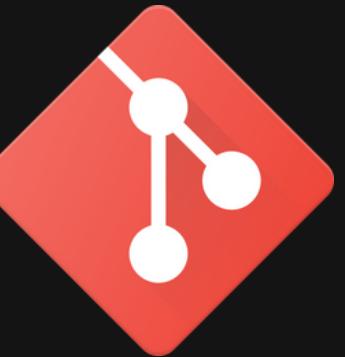
Foi desenvolvido pelo Linus Torvalds (desenvolvedor do Linux)

Criado em 2005 para gerenciar o desenvolvimento dos códigos do Linux

O GIT É TIPO UMA MÁQUINA  
DO TEMPO PRO SEU CÓDIGO.



# PORQUE USAR GIT?

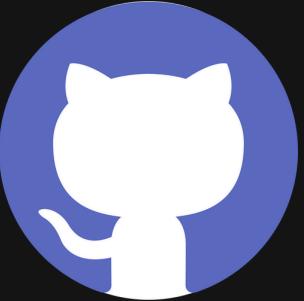


- Controle de versão – volta a qualquer versão do código.
- Trabalho colaborativo – várias pessoas trabalhando simultaneamente.
- Backup seguro – código salvo local e na nuvem.
- Histórico detalhado e rastreável – quem fez, quando e por quê.
- Integração com outras ferramentas – CI/CD e deploy automático.



# INTRODUZINDO GITHUB

---



O GitHub é uma plataforma de hospedagem de código-fonte e colaboração que utiliza o Git para controle de versão. É o maior repositório de código do mundo, onde milhões de desenvolvedores colaboram em projetos open-source e privados.

O GIT HUB É ONDE SE  
HOSPEDA OS REPOSITÓRIOS



# AMBIENTE DO GITHUB

The screenshot shows the GitHub Home page for the user `levyrodrigues23`. The left sidebar lists the user's repositories, including `DalyGames`, `Trabalho_Gerencia_ProfLa  
risse`, `Portifolio-CapacitaBr`, `resolve_leetcode`, `ENGEDADOS/Code-Tutor`, `trabalho_renata`, and the user's profile repository `levyrodrigues23/levyrodrigues23`. A green button labeled "New" is visible next to the repository list.

The main content area features a "Home" section with several calls-to-action: "Ask Copilot", "Get code feedback", "Create an issue for a bug", and "Create a profile README for me". Below this is a "Feed" section showing a recent contribution from `NelsonFelipe` to the repository `todo-react`. The contribution was merged 4 hours ago and involved 3 commits. A "Summary by CodeRabbit" section provides a detailed breakdown of the changes, mentioning overhauling CI, running tests across multiple Node.js versions, using build artifacts for deployment, and migrating deploy. A "Read more" link is provided for further details.

To the right, there is a "Learn. Collaborate. Grow." box for GitHub Education, which offers tools and community support for tech challenges. A "Go to GitHub Education" button is present. Another box titled "Latest changes" lists four recent updates: new features in GitHub Copilot in Eclipse, GitHub Actions AI labeler and moderator, a New Spark sharing option, and profile menu enhancements. A "View changelog →" link is at the bottom of this box.

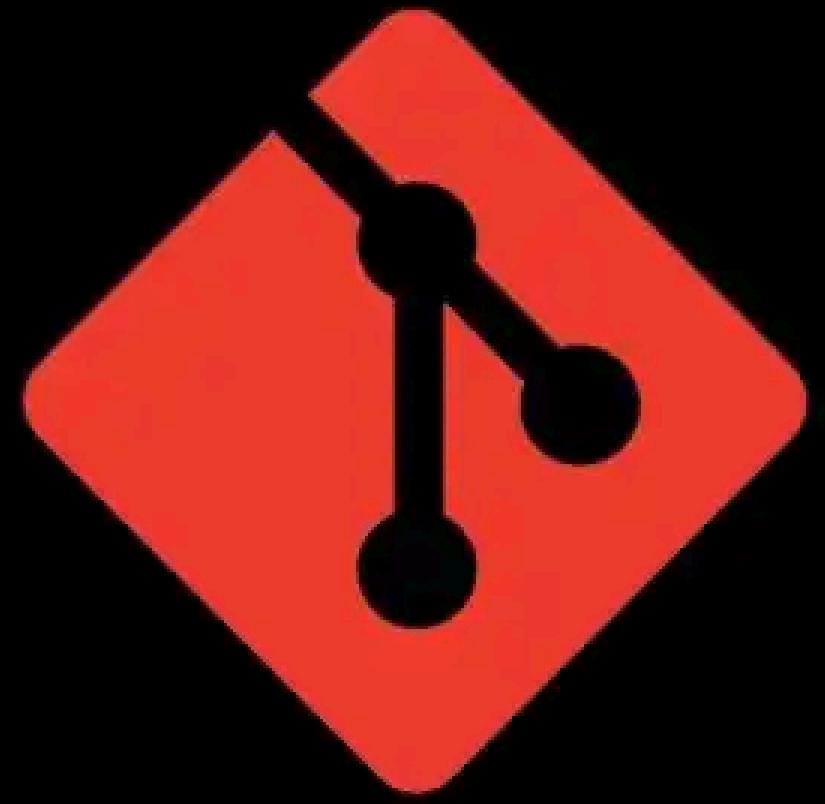


# PORQUE USAR GITHUB?

---

- Repositórios remotos – teu código fica disponível online, acessível de qualquer lugar.
- Colaboração facilitada – issues, pull requests, comentários, revisão de código.
- Comunidade e portfólio – mostrar projetos, contribuir em open source, fortalecer currículo.
- Integração com Git – sincroniza tuas branches e commits locais com o remoto.





git +



# INSTALAÇÃO DO GIT

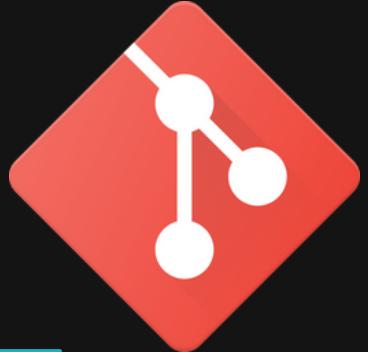


- Acessar a documentação do git
- Executar o processo de instalação a partir do sistema operacional da máquina

The screenshot shows the official Git website ([git-scm.com/](https://git-scm.com/)) with a dark theme. The top navigation bar includes a search bar and a gear icon. The main header features the Git logo and the tagline "local-branching-on-the-cheap". On the left, there's a sidebar with links for "About", "Documentation", "Downloads" (which is currently selected), and "Community". The "Downloads" section contains links for "macOS", "Windows", and "Linux/Unix". Below this, a note says "Older releases are available and the Git source repository is on GitHub." The right side of the page features a large monitor icon displaying the "Latest source Release 2.51.0" and a "Download for Windows" button. Other sections visible include "GUI Clients", "Logos", and "Git via Git".

# INSTALAÇÃO DO GIT

---



## 💻 WINDOWS

- Acesse o site oficial:

Vá para: <https://git-scm.com/downloads>

Clique em "Download for Windows"

O download começará automaticamente

- Execute o instalador:

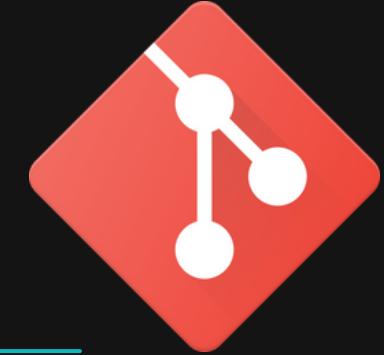
Rode o arquivo .exe baixado

Aceite as configurações padrão (são ideais para iniciantes)

Clique "Next" em todas as telas

- Na tela "Adjusting your PATH environment": deixe "Git from the command line and also from 3rd-party software"
- Finalize a instalação

# INSTALAÇÃO DO GIT



Git 2.15.1 for Windows

**Information**

Please read the following important information before continuing.

When you are ready to continue with Setup, click Next.

**12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.**

**END OF TERMS AND CONDITIONS**

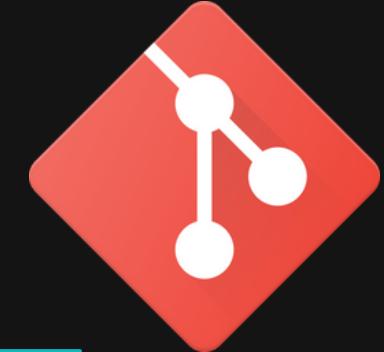
<https://gitforwindows.org/>

Refresh  Only show new options

Next Cancel

# INSTALAÇÃO DO GIT

---



```
# Atualize os pacotes  
sudo apt update
```

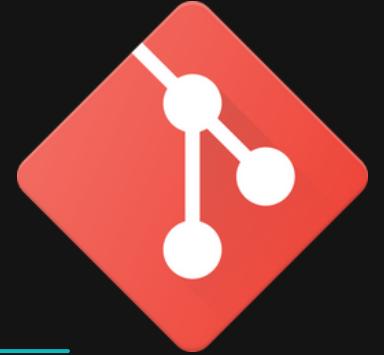
```
# Instale o Git  
sudo apt install git
```

Verifique a versão do git para ver se ele foi instalado

- `git --version`

# INSTALAÇÃO DO GIT

---



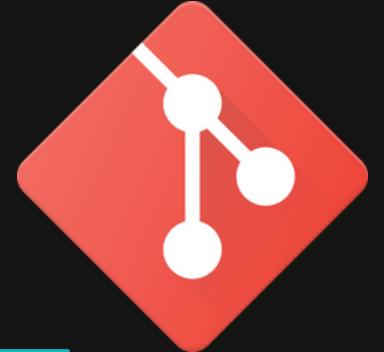
## POR QUE CONFIGURAR?

Antes de usar o Git, você PRECISA se identificar. Isso é obrigatório porque:

- Cada commit que você faz fica "assinado" com seu nome e email
- Colaboradores sabem quem fez cada mudança
- GitHub conecta seus commits ao seu perfil
- Histórico fica organizado e rastreável

# INSTALAÇÃO DO GIT

---



## ⚙️ CONFIGURAÇÕES OBRIGATÓRIAS

Use seu nome real (não nickname)

Coloque entre aspas se tiver espaços

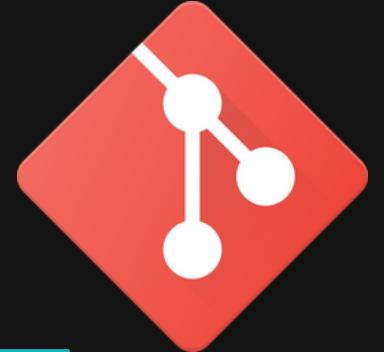
Será exibido no GitHub e em todos os commits



```
git config --global user.name "José Silva"
```

# INSTALAÇÃO DO GIT

---



## ⚙️ CONFIGURAÇÕES OBRIGATÓRIAS

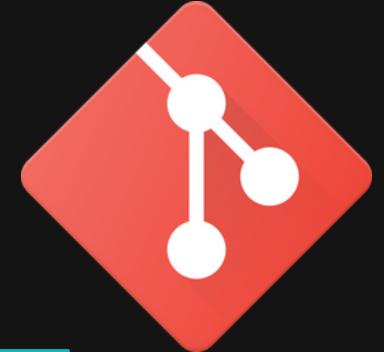
Use o mesmo email da sua conta GitHub,  
senão seus commits não aparecerão conectados ao seu perfil



```
git config --global user.email "jose.silva@email.com"
```

# INSTALAÇÃO DO GIT

---



VERIFICAR SE FUNCIONOU



```
git config --global user.name  
# Deve mostrar: José Silva
```

```
git config --global user.email  
# Deve mostrar: jose.silva@email.com
```



# Apagar credenciais

Com esse comando é possível apagar necessariamente todas as credenciais



```
git config --global --unset user.name  
git config --global --unset user.email
```

# Conceitos fundamentais

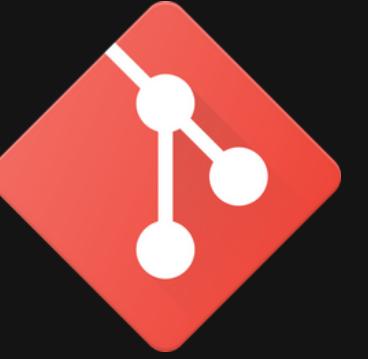
---



- Repositório (repo) → pasta/projeto versionado.
- Repositório Local → fica na tua máquina.
- Repositório Remoto → fica hospedado online(neste caso, no GitHub)
- Versão → estado salvo do projeto (histórico de mudanças).
- Controle de versão → sistema que registra todas as alterações feitas no código.
- Histórico → linha do tempo com todos os commits feitos no projeto.



**REPOSITÓRIO LOCAL ≠ REPOSITÓRIO REMOTO**



# Comandos básicos do git



# git init

Cria um repositório Git dentro da pasta atual. É como se você dissesse: “A partir de agora, essa pasta vai ser monitorada pelo Git”



git init

# git add

Prepara os arquivos para o próximo commit  
Move arquivos para a área de staging



git add arquivo.txt

git add . # adiciona todos os arquivos modificados

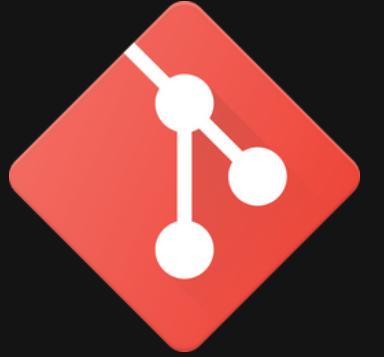


# git commit

Registra as mudanças no histórico do projeto com uma mensagem explicando o que foi feito



```
git commit -m "Adicionei a funcionalidade X"
```



# git status

Mostra o estado atual do repositório: quais arquivos foram modificados, quais estão no staging e quais não estão.



```
git status
```

# git remote



Ele serve para conectar o seu repositório local a um remoto, permitindo enviar (push) e receber (pull) alterações.



git remote add  
origin

---

# git log

Exibe o histórico de commits da branch atual, mostrando quem fez, quando fez e a mensagem do commit.



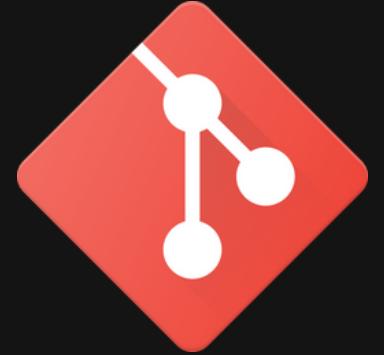
git log

# git config

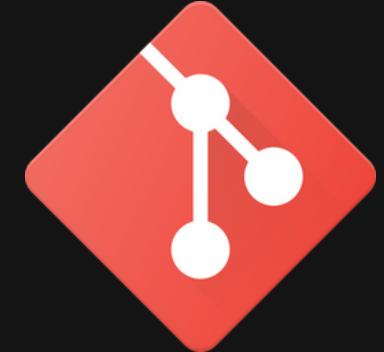
Define as configurações do Git, como nome e e-mail do autor dos commits. Normalmente feito só uma vez por máquina.



```
git config --global user.name "Seu Nome"  
git config --global user.email "seuemail@email.com"
```



# git push



Envia os commits locais para o repositório remoto



git push origin main



# git pull

Atualiza o repositório local trazendo as alterações do repositório remoto. É como um “download” das mudanças.



`git pull origin main`

# git clone

Clona o repositório remoto para a máquina local, trazendo consigo todos os arquivos possíveis (isso é meio óbvio já que é um clone)



`git clone https://github.com/levyrodrigues23/todo-react.git`

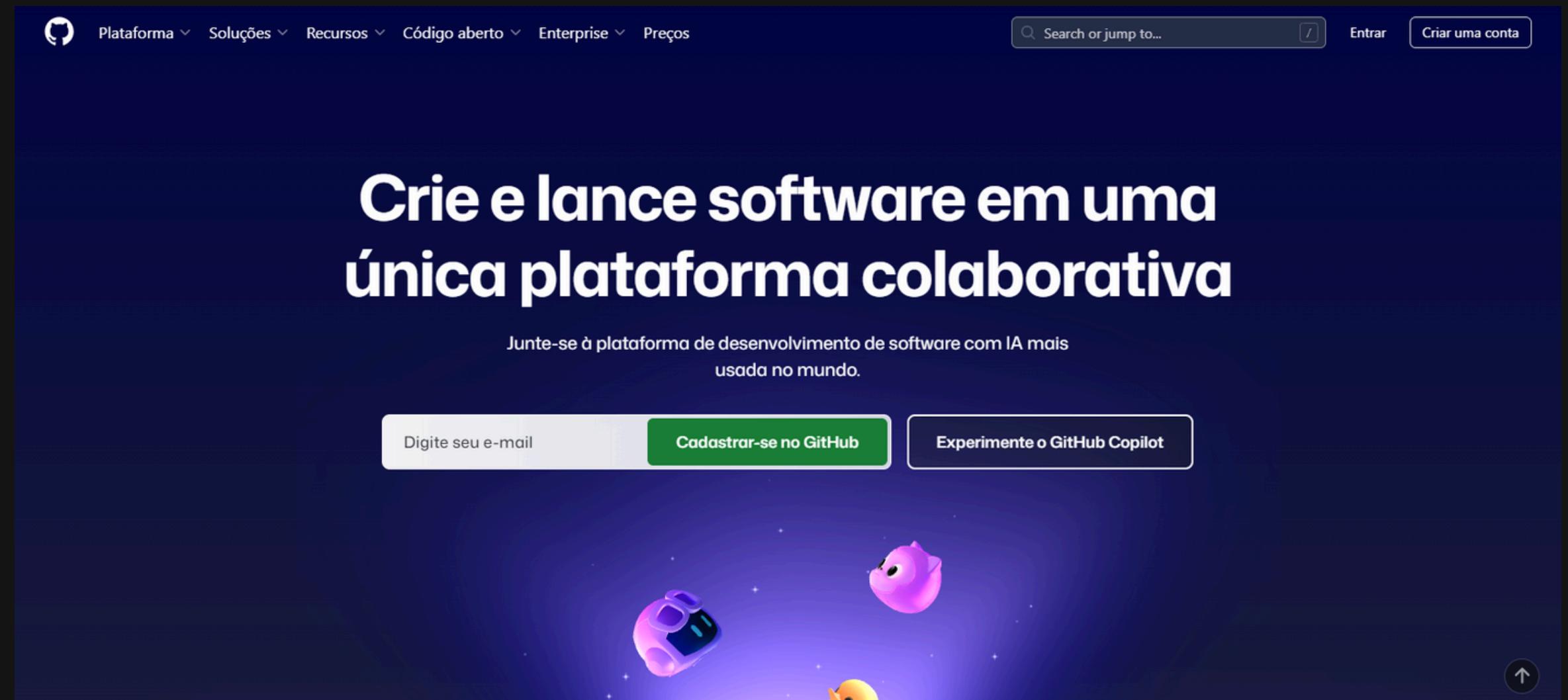
# GITHUB NA PRÁTICA

---



## CRIANDO SUA CONTA NO GITHUB

- Acesse:  
<https://github.com>
- Clique em "Sign up"
- Escolha um username único (será sua marca!)
- Use o mesmo email configurado no Git
- Senha forte e segura



# GITHUB NA PRÁTICA



- Criando repositórios no GitHub
- Adicionando um nome

The image shows two side-by-side screenshots of the GitHub interface. On the left, the GitHub Dashboard for the user 'levyrodrigues23' is displayed. It features a 'Top repositories' section, a search bar with the placeholder 'Find a repository...', and a prominent green 'New' button. On the right, the 'Create a new repository' form is shown. The 'General' tab is selected, showing the owner as 'levyrodrigues23' and a repository name field containing '/'. A note below suggests a short and memorable name like 'psychic-giggle'. The description field is currently empty, showing '0 / 350 characters'.

Dashboard

levyrodrigues23

Top repositories

New

Find a repository...

Create a new repository

Repositories contain a project's files and version history. Have a project elsewhere? [Import a repository](#).  
Required fields are marked with an asterisk (\*).

1 General

Owner \* Repository name \*

levyrodrigues23 /

Great repository names are short and memorable. How about **psychic-giggle**?

Description

0 / 350 characters



# GITHUB NA PRÁTICA

- Configurações do repositório e Criar repositório

The screenshot shows the 'Configuration' step of a GitHub repository creation process. It includes fields for visibility (set to 'Public'), template selection ('No template'), README addition ('Off'), .gitignore creation ('No .gitignore'), and license selection ('No license'). A green 'Create repository' button is at the bottom.

2 Configuration

**Choose visibility \***  
Choose who can see and commit to this repository

**Start with a template**  
Templates pre-configure your repository with files.

**Add README**  
READMEs can be used as longer descriptions. [About READMEs](#)

**Add .gitignore**  
.gitignore tells git which files not to track. [About ignoring files](#)

**Add license**  
Licenses explain how others can use your code. [About licenses](#)

**Create repository**



# GITHUB NA PRÁTICA

- Configuração do repositório na máquina local

**Quick setup — if you've done this kind of thing before**

[Set up in Desktop](#) or  [HTTPS](#)  [SSH](#) [https://github.com/levyrodrigues23/repositorio\\_teste.git](https://github.com/levyrodrigues23/repositorio_teste.git)

Get started by [creating a new file](#) or [uploading an existing file](#). We recommend every repository include a [README](#), [LICENSE](#), and [.gitignore](#).

**...or create a new repository on the command line**

```
echo "# repositorio_teste" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/levyrodrigues23/repositorio_teste.git
git push -u origin main

```

# COLABORAÇÃO NO GITHUB

---

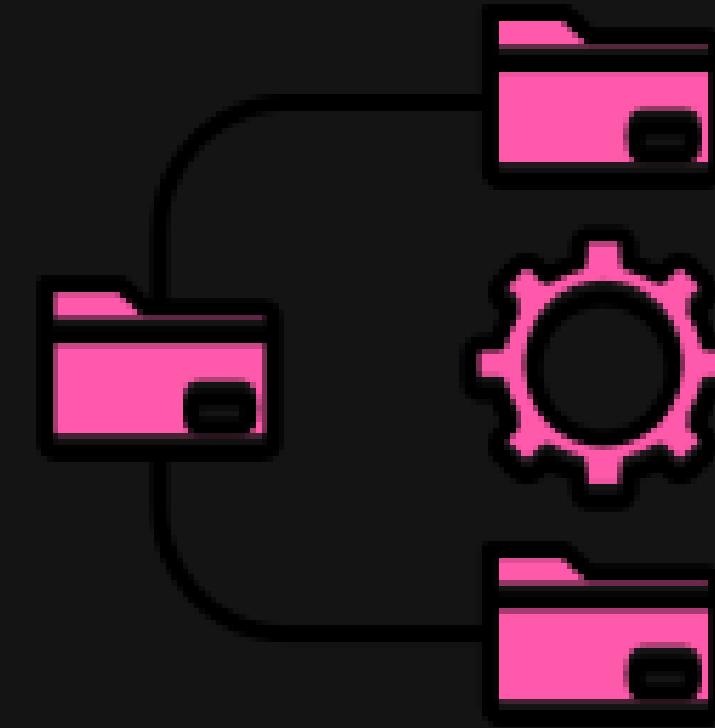
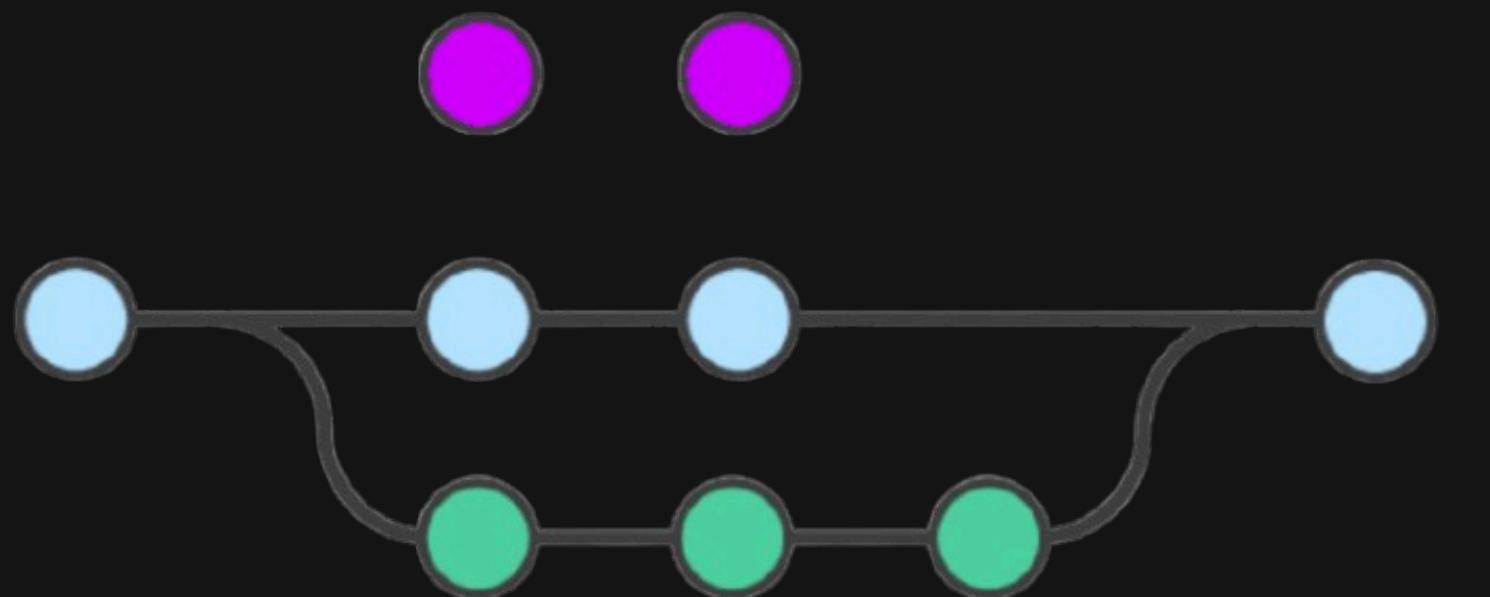


Colaboração no GitHub é quando várias pessoas trabalham juntas em um mesmo projeto de forma organizada, usando ferramentas como branches, pull requests e issues pra propor mudanças, revisar código e acompanhar tarefas.





# BRANCHES





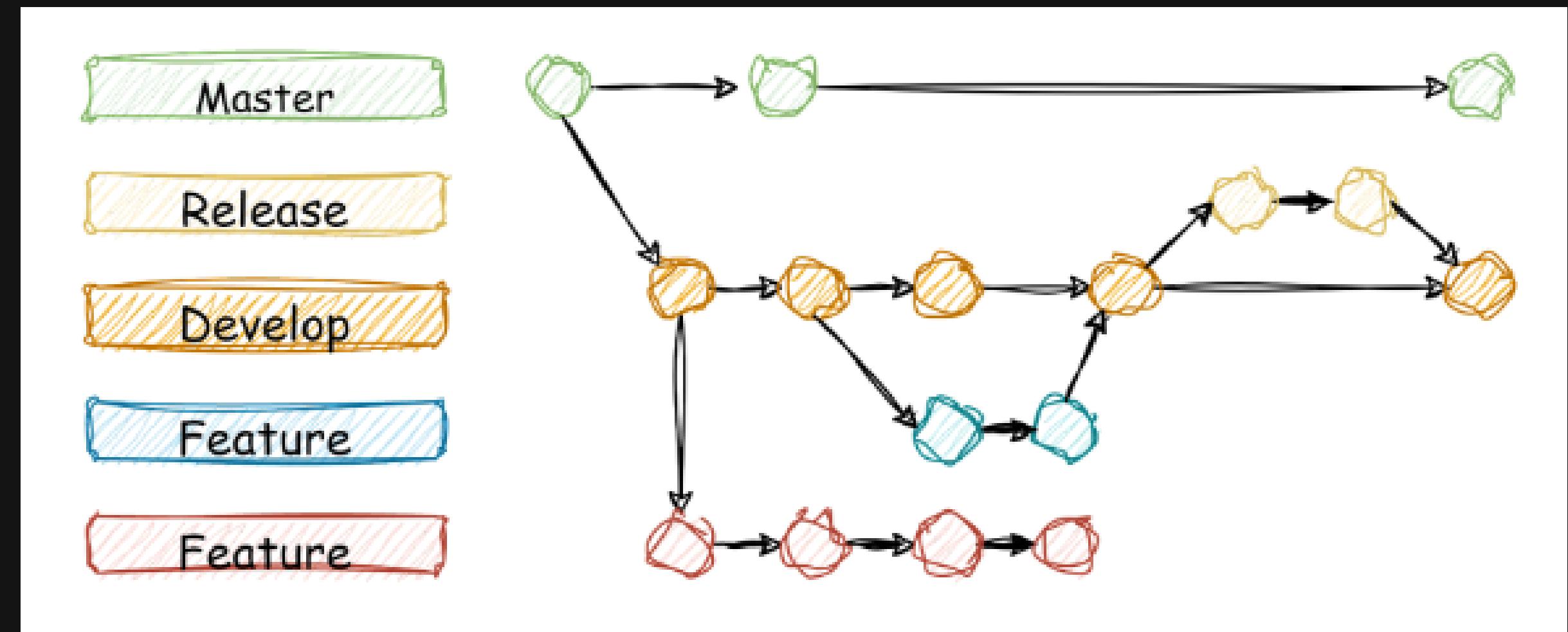
# O que são branches

É uma linha de desenvolvimento paralela e independente

Permite trabalhar em diferentes funcionalidades sem afetar o código principal

Como ter "várias versões" do seu projeto ao mesmo tempo

**BRANCH = "GALHO" OU  
"RAMIFICAÇÃO"**





# Por que usar Branches?

## Desenvolvimento Seguro

- Teste novas funcionalidades sem quebrar o código que já funciona
- Experimente sem medo - sempre pode voltar ao código estável

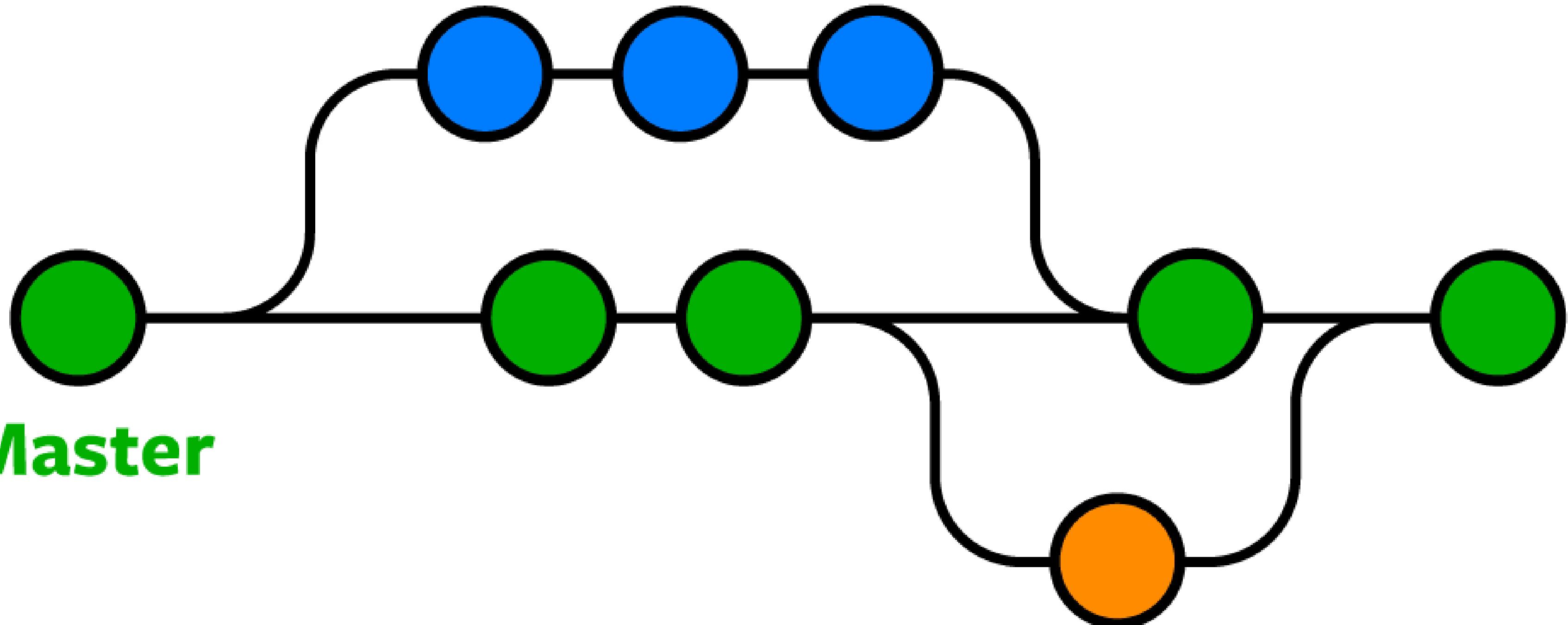
## Colaboração Eficiente

- Cada desenvolvedor trabalha em sua própria branch
- Evita conflitos entre membros da equipe

## Organização Clara

- Uma branch para cada funcionalidade/correção
- Histórico limpo e fácil de entender

# Your Work



Master

Someone Else's Work



# Comandos práticos

## Comandos Essenciais



```
git branch          # Lista todas as branches  
git branch nova-funcionalidade # Cria uma nova branch  
git checkout nova-funcionalidade # Muda para a branch  
git checkout -b nova-feature   # Cria E muda para a branch (atalho)  
git branch -d nome-branch      # Deleta uma branch
```



# Comandos práticos

1. Criando uma Nova Funcionalidade:



```
git checkout -b adicionar-login  
# Agora você está na branch "adicionar-login"
```



# Comandos práticos

## 2. Trabalhando na Branch:



```
# Faça suas modificações...
git add .
git commit -m "Adiciona tela de login"
git push origin adicionar-login
```

# Comandos práticos



3. Voltando para a Branch Principal:



```
git checkout main  
# Agora você está de volta na branch principal
```



# Comandos práticos

4. Quando sua funcionalidade está pronta:



```
git checkout main          # Vai para a branch principal  
git merge adicionar-login # Traz as mudanças da sua branch  
git branch -d adicionar-login # Deleta a branch (opcional)
```



# Boas práticas

O ato de usar **nomes descritivos** para as features que serão trabalhadas



- ✓ `git checkout -b adicionar-autenticacao-google`
- ✗ `git checkout -b nova-coisa`



# Boas práticas

Branches Pequenas e Focadas

Uma funcionalidade por branch

Merge frequente para evitar conflitos grandes

The screenshot shows a dark-themed GitHub interface. At the top, there's a header with a user icon, the word 'main' with a dropdown arrow, '6 Branches', and '0 Tags'. Below this is a modal titled 'Switch branches/tags' with a search bar containing 'Find or create a branch...'. The modal has two tabs: 'Branches' (which is active) and 'Tags'. A list of branches is shown, with 'main' being the default branch (indicated by a checkmark and a 'default' badge). Other branches listed include 'chore/update-ci-workflow', 'develop', 'docker-feature', 'dockerizando', and 'test/add-initial-tests'. At the bottom of the modal is a button labeled 'View all branches'.



# Boas práticas

## Uso do .gitignore

O .gitignore é responsável por guardar os arquivos que não serão expostos no repositório remoto



```
❶ .gitignore
1  # Logs
2  logs
3  *.log
4  npm-debug.log*
5  yarn-debug.log*
6  yarn-error.log*
7  pnpm-debug.log*
8  lerna-debug.log*
9
10 node_modules
11 dist
12 dist-ssr
13 *.local
14
15 # Editor directories and files
16 .vscode/*
17 !.vscode/extensions.json
18 .idea
19 .DS_Store
20 *.suo
21 *.ntvs*
22 *.njsproj
23 *.sln
24 *.sw?
25
26 .vercel
27
```



# Boas práticas

README.md bem escrito,  
destacando todo passo-a-  
passo do projeto

README

---

**todo-react**

**Todo React**

Este é um projeto de lista de tarefas (Todo List) desenvolvido com React. Ele tem como objetivo oferecer uma interface simples e intuitiva para organizar e gerenciar suas tarefas diárias.

**Funcionalidades**

- Adicionar novas tarefas
- Marcar tarefas como concluídas
- Remover tarefas
- Visualizar lista de tarefas pendentes e concluídas
- Interface responsiva e moderna

**Tecnologias Utilizadas**

- [React](#)
- [JavaScript](#)
- [CSS](#)

**Como Executar o Projeto**

1. Clone o repositório:



# GITHUB NA PRÁTICA

---

- **Issues**

São como chamados/tarefas dentro do repositório. Dá pra usar pra relatar bugs, sugerir melhorias ou organizar o que precisa ser feito. É tipo um quadro de “pendências” do projeto.

- **Pull Requests**

É quando alguém faz uma mudança no código (numa branch) e pede pra essa mudança ser revisada e juntada (merge) na branch principal. Serve pra discutir, revisar e aprovar código antes de ir pro projeto principal.

# GITHUB NA PRÁTICA



Acessando as issues dentro do projeto

The screenshot shows a GitHub project page for 'levyrodrigues23 / todo-react'. The top navigation bar includes links for Code, Issues (which is highlighted with a red box), Pull requests, Actions, Projects, Wiki, Security, Insights, and Settings. Below the navigation is a search bar with the query 'is:issue state:open' and buttons for Labels, Milestones, and New issue. A filter bar at the bottom allows filtering by Open (0) or Closed (0) issues, and provides dropdowns for Author, Labels, Projects, Milestones, Assignees, and sorting by Newest. The main content area displays the message 'No results' with the sub-instruction 'Try adjusting your search filters.'

# GITHUB NA PRÁTICA



## Adicionar filtro de tarefas por prioridade #12

Open



levyrodrigues23 opened 3 minutes ago

Owner ...

Seria interessante implementar uma funcionalidade que permite ao usuário filtrar as tarefas pela prioridade definida (alta, média, baixa). Isso facilitaria a organização e o foco nas tarefas mais importantes.

Sugestão de funcionalidade:

- Adicionar um seletor de prioridade na interface de tarefas
- Permitir filtrar e exibir apenas tarefas de uma determinada prioridade
- Atualizar a documentação e exemplos, se necessário

Benefícios:

- Melhora a usabilidade
- Facilita a gestão das tarefas

Caso precise de mais detalhes ou exemplos, posso complementar.

Create sub-issue



## Issue criada

# GITHUB NA PRÁTICA



## Acessando os pull requests

The screenshot shows a GitHub repository page for 'levyrodrigues23 / todo-react'. The top navigation bar includes links for Code, Issues (with 1), Pull requests (highlighted with a red box), Actions, Projects, Wiki, Security, Insights, and Settings. Below the navigation is a search bar with filters for 'is:pr is:open' and buttons for Labels (9), Milestones (0), and New pull request. At the bottom, there are statistics for 0 Open and 11 Closed pull requests, along with dropdown menus for Author, Label, Projects, Milestones, Reviews, Assignee, and Sort.

OBS: Para criar um pull request, voce precisa de outras branches para poder fazer as comparações

# GITHUB NA PRÁTICA



Aqui voce pode criar um pull request antes de fazer o merge das alterações perante a comparação das branches.

Add a title

Merge pull request #11 from levyrodrigues23/develop

Add a description

Write Preview

Develop

Markdown is supported

Paste, drop, or click to add files

Create pull request ▾

Remember, contributions to this repository should follow our [GitHub Community Guidelines](#).

A screenshot of a GitHub pull request creation interface. It shows a title field containing "Merge pull request #11 from levyrodrigues23/develop", a description field containing "Develop", and a rich text editor toolbar above them. Below the editor is a file upload area with the placeholder "Paste, drop, or click to add files". At the bottom right is a green "Create pull request" button with a dropdown arrow. A note at the bottom states: "Remember, contributions to this repository should follow our [GitHub Community Guidelines](#)".

# GITHUB NA PRÁTICA

---



Após o processo anterior, é possível realizar o merge das alterações :)

A screenshot of a GitHub pull request interface. At the top left is a green icon with two hands holding a chain. To its right, a green circle contains a white checkmark. Next to it is the text "All checks have passed" and "7 successful checks". Below this is another green circle with a white checkmark, followed by the text "No conflicts with base branch" and "Merging can be performed automatically." At the bottom left is a large green button with the text "Merge pull request" and a dropdown arrow. To the right of the button is the text "You can also merge this with the command line. [View command line instructions.](#)"

All checks have passed  
7 successful checks

No conflicts with base branch  
Merging can be performed automatically.

Merge pull request ▾ You can also merge this with the command line. [View command line instructions.](#)

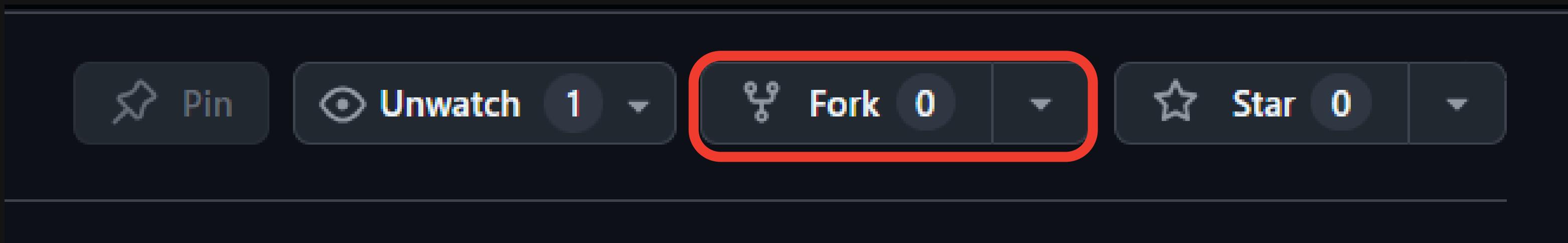
# GITHUB NA PRÁTICA

---



- Fork

Um fork é essencialmente uma cópia de um repositório de código que permite desenvolvimento independente. Quando você faz um fork, está criando sua própria versão do projeto onde pode fazer modificações sem afetar o código original.



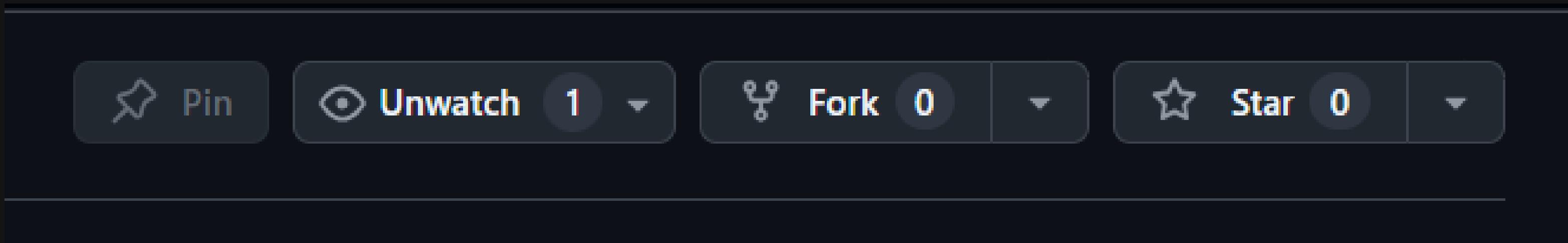
# GITHUB NA PRÁTICA

---



- Fork

Um fork é essencialmente uma cópia de um repositório de código que permite desenvolvimento independente. Quando você faz um fork, está criando sua própria versão do projeto onde pode fazer modificações sem afetar o código original.



# GITHUB NA PRÁTICA



Cria uma cópia do repositório na tua conta do GitHub, que tu pode modificar à vontade e depois propor mudanças pro original via pull request.

## Create a new fork

A *fork* is a copy of a repository. Forking a repository allows you to freely experiment with changes without affecting the original project.

*Required fields are marked with an asterisk (\*).*

Owner \*

Choose an owner ▾

Repository name \*

todo-react

By default, forks are named the same as their upstream repository. You can customize the name to distinguish it further.

Description (optional)

deploy

Copy the main branch only

Contribute back to levyrodrigues23/todo-react by adding your own branch. [Learn more.](#)

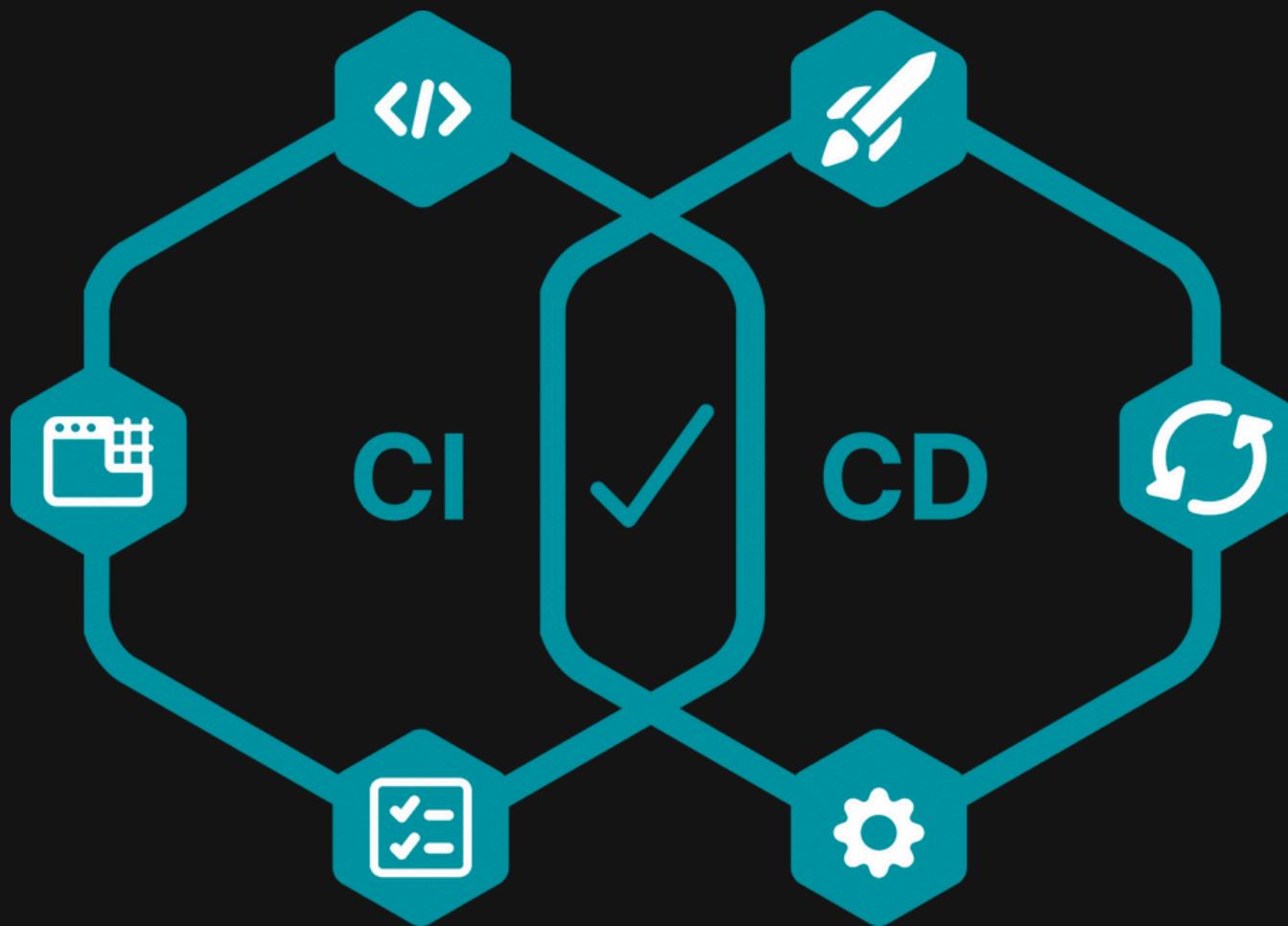
## GIT CLONE ≠ FORK

[Create fork](#)

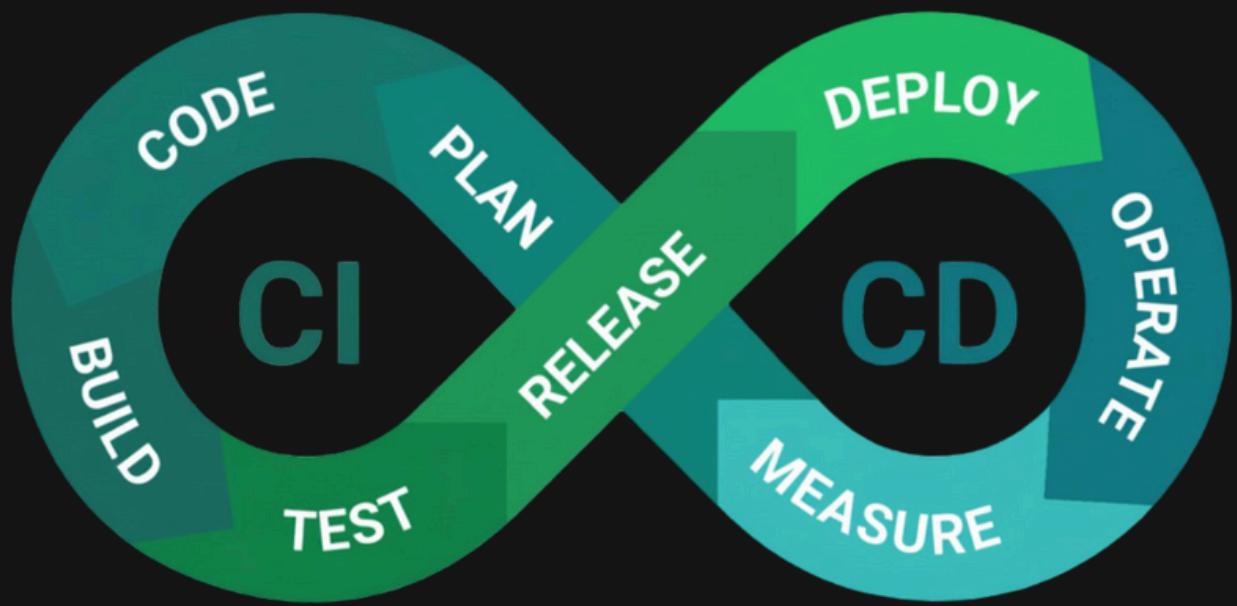
# INTEGRAÇÃO CONTÍNUA (CI)

&

# ENTREGA CONTÍNUA (CD)



# O que é CI/CD?



CI/CD é uma prática essencial no desenvolvimento de software moderno que visa **automatizar** as etapas durante o desenvolvimento de software. Dessa forma, as equipes de desenvolvimento conseguem entregar software de forma mais rápida, segura e confiável.

# OBJETIVOS E VANTAGENS

---

## CI

- Detectar problemas mais cedo
- Reduzir conflitos de merge
- Melhorar a qualidade do código

## CD

- Lançamentos de baixo risco
- Entregas mais rápidas
- Feedback rápido

# INTEGRAÇÃO CONTÍNUA (CI)

---

A Integração Contínua é a parte do processo focada nos **desenvolvedores** e na **qualidade do código**. A ideia principal é integrar o código de múltiplos desenvolvedores em um repositório central. Ele foca em garantir que o novo código não "quebre" o que já existe e está funcionando.

# ENTREGA CONTÍNUA (CD)

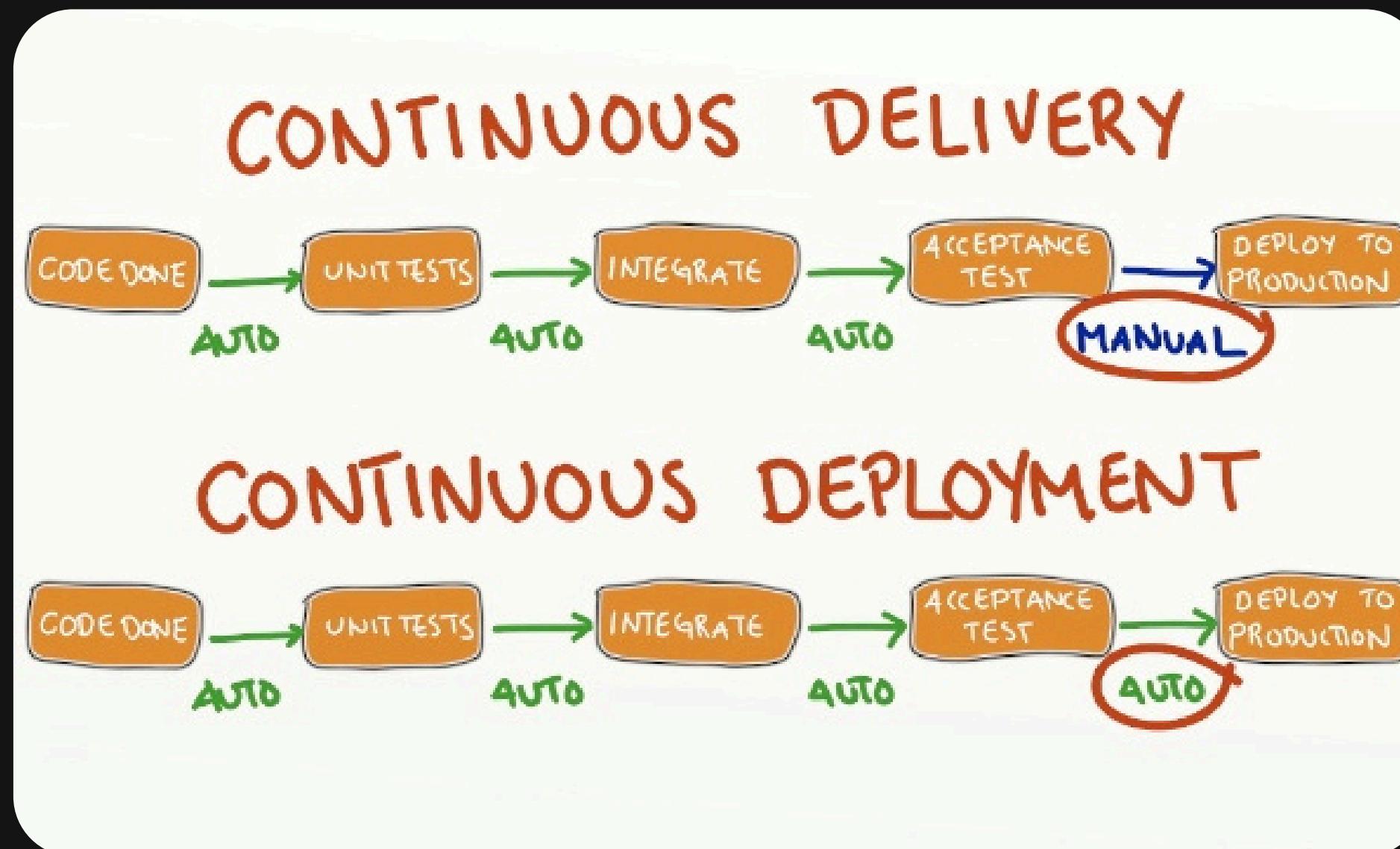
---

Automatiza o processo de **lançamento** do software para os ambientes de teste e produção. Isso permite que a equipe de negócios ou de qualidade decida o melhor momento para liberar a nova versão para os usuários finais. O CD ocorrem em 2 partes: Entrega Contínua e Implantação Contínua.

O CD ocorre após o CI, caso tenha passado nas etapas de (build e teste).

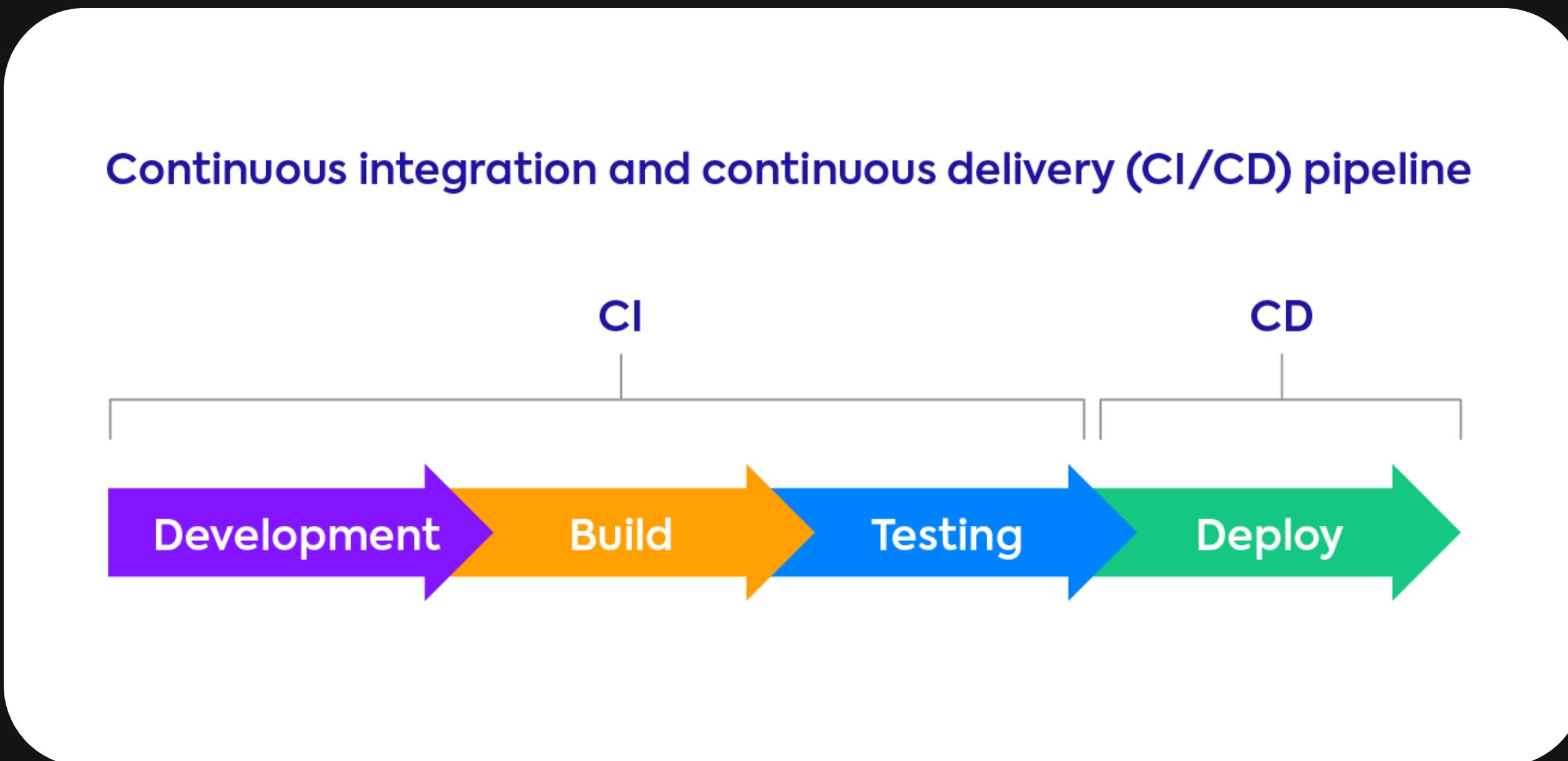
- **ENTREGA CONTÍNUA: O PROCESSO É AUTOMATIZADO ATÉ A PRODUÇÃO. O LANÇAMENTO FINAL É MANUAL.**
- **IMPLEMENTAÇÃO CONTÍNUA: SE O CÓDIGO PASSAR POR TODOS OS TESTES AUTOMATIZADOS, ELE É LANÇADO EM PRODUÇÃO AUTOMATICAMENTE, SEM QUALQUER INTERVENÇÃO HUMANA.**

# ENTREGA CONTÍNUA (CD)

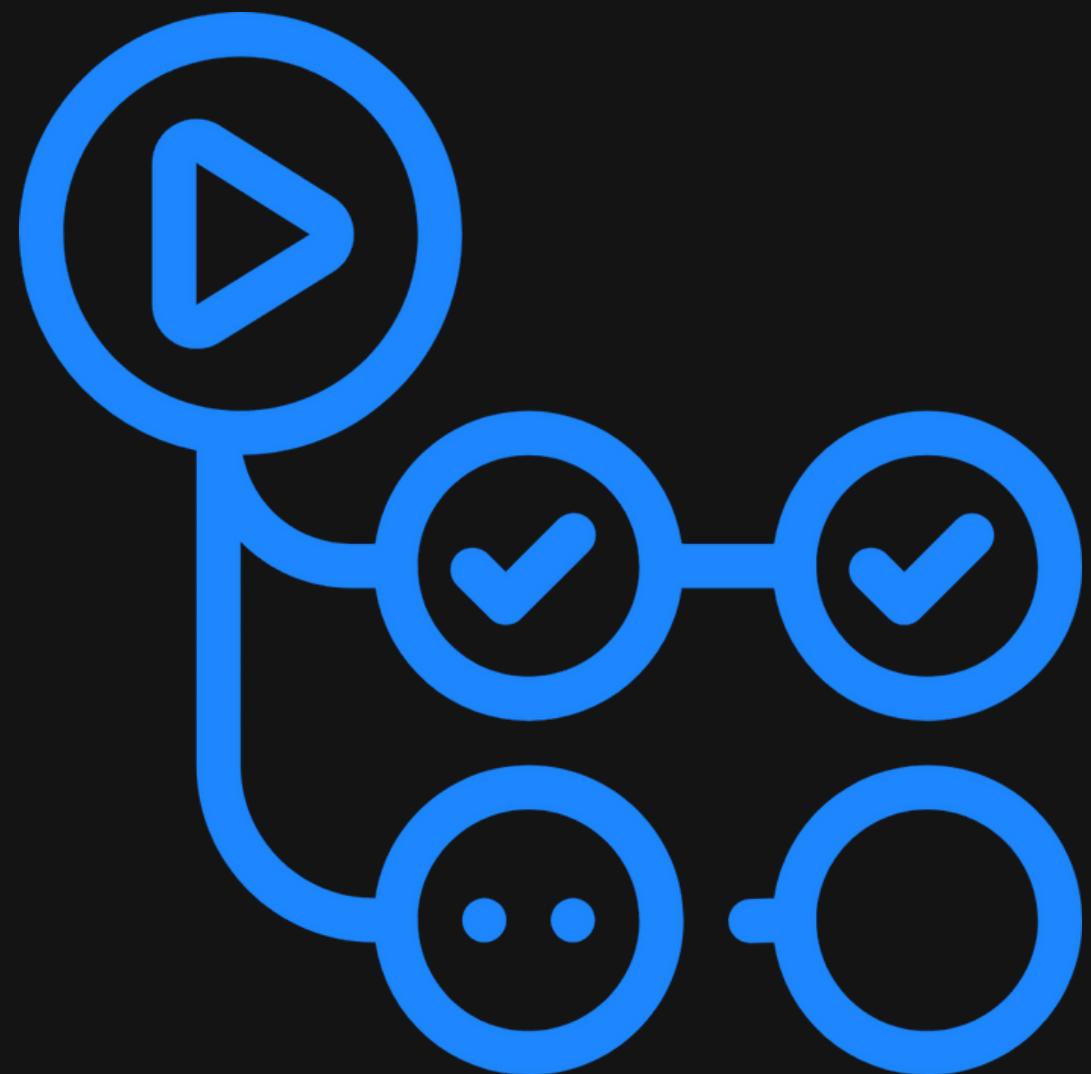


# FLUXO DE CI/CD

---



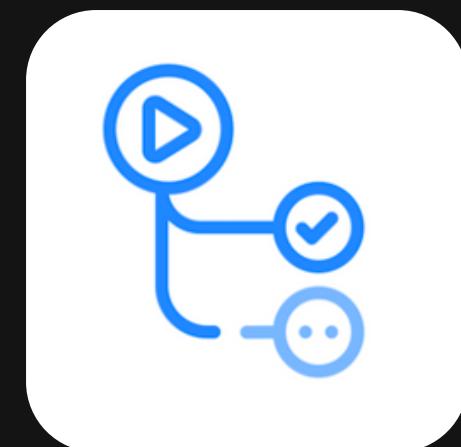
# CI/CD COM GITHUB ACTIONS



# GITHUB ACTIONS

---

GitHub Actions é uma [plataforma](#) de automação integrada ao GitHub, com ela podemos automatizar uma vasta gama de tarefas e outras operações diretamente dos repositórios. Em essência, o GitHub Actions funciona como uma ferramenta de Integração Contínua e Entrega Contínua (CI/CD).



# GITHUB ACTIONS

The screenshot shows a GitHub repository page for a private repository named "To-do-list-for-DevOps-concepts". The repository has 2 branches and 0 tags. The commit history shows a merge pull request from NelsonFelipe/develop, which included setting base code for files like public, src, .gitignore, LICENSE, README.md, eslint.config.js, index.html, package-lock.json, package.json, and vite.config.js. The commits were made 24 minutes ago. The repository has 0 stars, 0 forks, and 0 watching.

NelsonFelipe / To-do-list-for-DevOps-concepts

Type / to search

Code Issues Pull requests Actions Projects Security Insights Settings

To-do-list-for-DevOps-concepts Private

Watch 0 Fork 0 Star 0

main 2 Branches 0 Tags

Go to file Add file Code About

NelsonFelipe Merge pull request #1 from NelsonFelipe/develop 7b922e5 · 3 hours ago 3 Commits

public setting base code 24 minutes ago

src setting base code 24 minutes ago

.gitignore setting base code 24 minutes ago

LICENSE Initial commit 4 hours ago

README.md setting base code 24 minutes ago

eslint.config.js setting base code 24 minutes ago

index.html setting base code 24 minutes ago

package-lock.json setting base code 24 minutes ago

package.json setting base code 24 minutes ago

vite.config.js setting base code 24 minutes ago

Project used to apply basic concepts about DevOps.

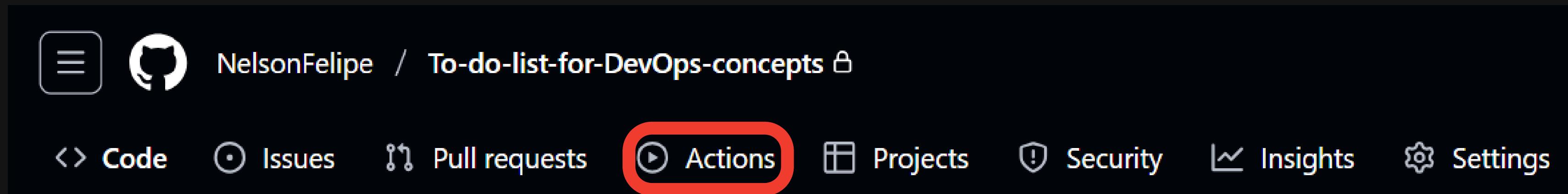
Readme MIT license Activity 0 stars 0 watching 0 forks

Releases No releases published Create a new release

Packages No packages published Publish your first package

# GITHUB ACTIONS

---



# GITHUB ACTIONS

The screenshot shows the GitHub Actions interface for the repository `levyrodrigues23 / todo-react`. The top navigation bar includes links for Code, Issues, Pull requests, Actions (which is the active tab), Projects, Wiki, Security, Insights, and Settings. A search bar and user profile icons are also present.

**Get started with GitHub Actions**

Build, test, and deploy your code. Make code reviews, branch management, and issue triaging work the way you want. Select a workflow to get started.

Skip this and set up a workflow yourself →

Search workflows

**Suggested for this repository**

- Publish Node.js Package to GitHub Packages** By GitHub Actions Publishes a Node.js package to GitHub Packages. [Configure](#) JavaScript ●
- Jekyll using Docker image** By GitHub Actions Package a Jekyll site using the `jekyll/builder` Docker image. [Configure](#) HTML ●
- Publish Node.js Package** By GitHub Actions Publishes a Node.js package to npm. [Configure](#) JavaScript ●
- Grunt** By GitHub Actions Build a NodeJS project with npm and grunt. [Configure](#) JavaScript ●
- Gulp** By GitHub Actions Build a NodeJS project with npm and gulp. [Configure](#) JavaScript ●
- Webpack** By GitHub Actions Build a NodeJS project with npm and webpack. [Configure](#) JavaScript ●

**Deployment**

[View all](#)

- Deploy Node.js to Azure Web App** By Microsoft Azure Build a Node.js project and deploy it to an Azure Web App. [Configure](#) Deployment ●
- Deploy Node.js to Azure Functions App** By Microsoft Azure Build a Node.js project and deploy it to an Azure Functions App on Windows or Linux. [Configure](#) Deployment ●
- Deploy to Amazon ECS** By Amazon Web Services Deploy a container to an Amazon ECS service powered by AWS Fargate or Amazon EC2. [Configure](#) Deployment ●
- Build and Deploy to GKE** By Google Cloud Build a docker container, publish it to Google Container Registry, and deploy to GKE. [Configure](#) Deployment ●

# GITHUB ACTIONS

The screenshot shows the GitHub Actions interface for the repository 'levyrodrigues23/todo-react'. The top navigation bar includes links for Code, Issues, Pull requests, Actions (which is the active tab), Projects, Wiki, Security, Insights, and Settings. A search bar and user profile icons are also present.

The main section is titled 'Choose a workflow' with the sub-instruction: 'Build, test, and deploy your code. Make code reviews, branch management, and issue triaging work the way you want. Select a workflow to get started.' Below this, there is a link to 'Skip this and set up a workflow yourself →'.

A sidebar on the left lists categories: Deployment, Security, Continuous integration, Automation, and Pages. A search bar at the top right contains the text 'Node.js'.

The central area displays a grid of workflow cards. A red box highlights the first card:

- Node.js** By GitHub Actions
- Build and test a Node.js project with npm.
- Configure JavaScript

Other visible cards include:

- Publish Node.js Package to GitHub Packages** By GitHub Actions
- Publish Node.js Package** By GitHub Actions
- Deploy Node.js to Azure Web App** By Microsoft Azure
- Deploy Node.js to Azure Functions App** By Microsoft Azure
- njsscan** By NodeJSScan
- Red Hat CodeReady Dependency Analytics** By Red Hat

# GITHUB ACTIONS

The screenshot shows the GitHub Actions workflow editor for a repository named 'todo-react'. The workflow file is 'node.js.yml' located in the 'main' branch. The code defines a workflow named 'Node.js CI' that runs on pushes to the 'main' branch. It uses the 'ubuntu-latest' runner and a matrix strategy with Node.js versions 18.x, 20.x, and 22.x. The workflow includes steps for checkout, setting up Node.js, running npm ci, building, and testing.

The GitHub Marketplace sidebar is open, displaying featured actions:

- Cache** by actions (5k stars): Cache artifacts like dependencies and build outputs to improve workflow execution time.
- Setup Node.js environment** by actions (4.4k stars): Setup a Node.js environment by adding problem matchers and optionally downloading and adding it to the PATH.
- Setup Java JDK** by actions (1.7k stars): Set up a specific version of the Java JDK and add the command-line tools to the PATH.
- Download a Build Artifact** by actions (1.7k stars): Download a build artifact that was previously uploaded in the workflow by the upload-artifact action.
- Setup Go environment** by actions (1.6k stars): Setup a Go environment and add it to the PATH.

Below the featured actions are featured categories:

- Code quality
- Monitoring
- Continuous integration
- Project management
- Deployment
- Testing

At the bottom of the page, there are keyboard usage instructions:

- Use **Control + Shift + m** to toggle the **tab** key moving focus. Alternatively, use **esc** then **tab** to move to the next interactive element on the page.
- Use **Control + Space** to trigger autocomplete in most situations.

# GITHUB ACTIONS

**ON:** DEFINE OS GATILHOS QUE ACIONAM OS WORKFLOWS

**JOB:** CONTÉM AS "TAREFAS" QUE SERÃO REALIZADAS PELO WORKFLOW.

**RUNS-ON:** DEFINE O AMBIENTE VIRTUAL (VM) ONDE OS PASSOS SERÃO EXECUTADOS.

**STRATEGY:** CONTROLA COMO O JOB SERÁ EXECUTADO EM DIFERENTES VARIAÇÕES, GERALMENTE COM UMA MATRIZ (MATRIX).

**STEPS:** DEFINEM AS INSTRUÇÕES INDIVIDUAIS DENTRO DE UM JOB. SÃO EXECUTADOS EM ORDEM.

**USES:** DEFINEM O USO DE UMA ACTION

**NAME:** DEFINE UM RÓTULO

**WITH:** DEFINE INPUTS PARA AS ACTIONS.

**CACHE:** ACELERA EXECUÇÕES FUTURAS.

**RUN:** EXECUTAR COMANDOS DE SHELL.

```
4   name: Node.js CI
5
6   ▼ on:
7     ▼ push:
8       ··· branches: [ "main" ]
9     ▼ pull_request:
10       ··· branches: [ "main" ]
11
12   ▼ jobs:
13     ▼ build:
14
15       ··· runs-on: ubuntu-latest
16
17       ··· strategy:
18       ··· matrix:
19         ··· node-version: [ 18.x, 20.x, 22.x ]
20         ··· # See supported Node.js release schedule at https://nodejs.org/en/about/releases/
21
22       ··· steps:
23         - uses: actions/checkout@v4
24     ▼ - name: Use Node.js ${{ matrix.node-version }}
25       ··· uses: actions/setup-node@v4
26     ▼ - with:
27       ··· node-version: ${{ matrix.node-version }}
28       ··· cache: 'npm'
29         - run: npm ci
30         - run: npm run build --if-present
31         - run: npm test
32
```

**todo-react / .github / workflows /**

# GITHUB ACTIONS

---

```
14   - name: Lint
15     runs-on: ubuntu-latest
16     steps:
17       - name: Checkout repository
18         uses: actions/checkout@v4
19       - name: Setup Node.js
20         uses: actions/setup-node@v4
21         with:
22           node-version: '20'
23           cache: 'npm'
24       - name: Install dependencies
25         run: npm ci
26       - name: Run ESLint
27         run: npm run lint
```

# GITHUB ACTIONS

---

```
29   - test:
30     - runs-on: ubuntu-latest
31     - strategy:
32       - matrix:
33         - node-version: [18.x, 20.x, 22.x]
34       - steps:
35         - uses: actions/checkout@v4
36         - name: Use Node.js ${{ matrix.node-version }}
37         - uses: actions/setup-node@v4
38         - with:
39           - node-version: ${{ matrix.node-version }}
40           - cache: 'npm'
41         - run: npm ci
42         - run: npm test
```

# GITHUB ACTIONS

---

```
44   - build:
45     - runs-on: ubuntu-latest
46     - needs: test
47     - steps:
48       - uses: actions/checkout@v4
49       - uses: actions/setup-node@v4
50       - with:
51         - node-version: '20'
52         - cache: 'npm'
53         - run: npm ci
54         - run: npm run build --if-present
55         - name: Upload artifact
56         - uses: actions/upload-artifact@v4
57         - with:
58           - name: production-files
59           - path: ./dist
```

# GITHUB ACTIONS

NEEDS: [BUILD, LINT, TEST]

IF: GITHUB.REF == 'REFS/HEADS/MAIN'

FIREBASE\_TOKEN: \${{ SECRETS.FIREBASE\_TOKEN }}

```
61   ·   deploy:
62     ·   ·   runs-on: ubuntu-latest
63     ·   ·   needs: [build, lint, test]
64     ·   ·   if: github.ref == 'refs/heads/main'
65     ·   ·   steps:
66       - name: Checkout repository
67         uses: actions/checkout@v4
68       - name: Setup Node.js
69         uses: actions/setup-node@v4
70         with:
71           node-version: '20'
72           cache: 'npm'
73       - name: Install dependencies and Build
74         run:
75           npm ci
76           npm run build --if-present
77       - name: Deploy to Firebase Hosting
78         uses: w9jds/firebase-action@master
79         with:
80           args: deploy --only hosting
81         env:
82           FIREBASE_TOKEN: ${{ secrets.FIREBASE_TOKEN }}
```

# GITHUB ACTIONS

Code Issues ① Pull requests Actions Projects Wiki Security Insights Settings

← Node.js CI Merge pull request #11 from levyrodrigues23/develop #19 Re-run all jobs ...

**Summary**

Triggered via push 1 hour ago Status Success Total duration 1m 17s Artifacts 1

NelsonFelipe pushed → 6ea05d4 main

Jobs

- lint
- test (18.x)
- test (20.x)
- test (22.x)
- build
- deploy

node.js.yml on: push

Matrix: test

3 jobs completed

Show all jobs

build 14s deploy 34s

lint 15s

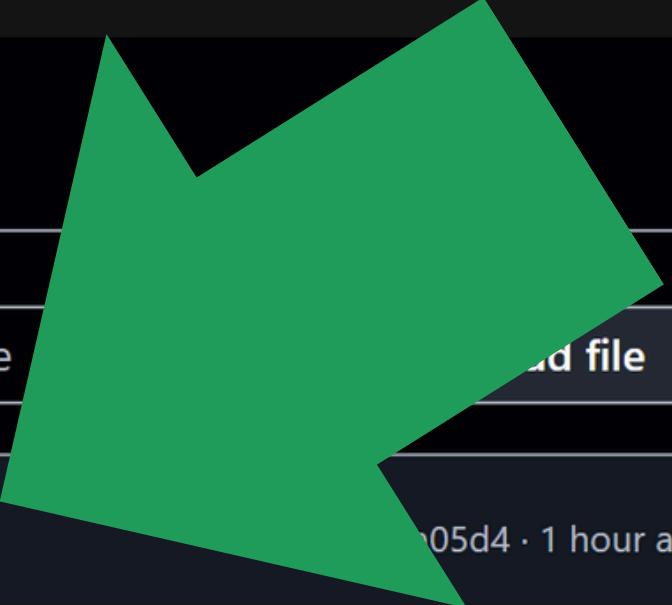
Run details Usage Workflow file

Artifacts Produced during runtime

Name	Size	Digest
production-files	79.3 KB	sha256:8aca3e5d4f814019681cb3f0d2ed0da26880901b6367aa3cb46c...

[+]

# GITHUB ACTIONS



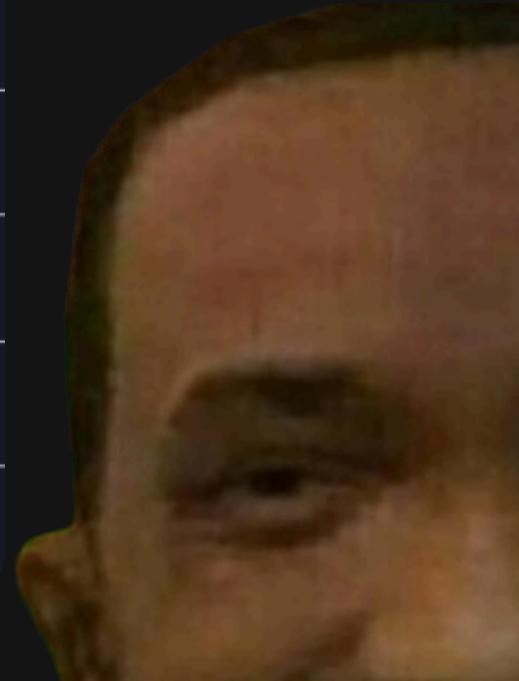
A screenshot of a GitHub repository page for "todo-react". The page shows 8 branches and 0 tags. A recent merge pull request from NelsonFelipe has been merged, with a timestamp of 2023-05-04 · 1 hour ago and 24 commits. The interface includes standard GitHub navigation buttons like "main", "Code", and search.



All checks have passed

7 successful checks

✓  Node.js CI / lint (push)	Successful in 15s	<a href="#">Details</a>
✓  Node.js CI / test (18.x) (push)	Successful in 19s	<a href="#">Details</a>
✓  Node.js CI / test (20.x) (push)	Successful in 12s	<a href="#">Details</a>
✓  Node.js CI / test (22.x) (push)	Successful in 12s	<a href="#">Details</a>
✓  Node.js CI / build (push)	Successful in 14s	<a href="#">Details</a>
✓  Node.js CI / deploy (push)	Successful in 34s	<a href="#">Details</a>
✓  CodeRabbit - Review skipped		<a href="#">Details</a>

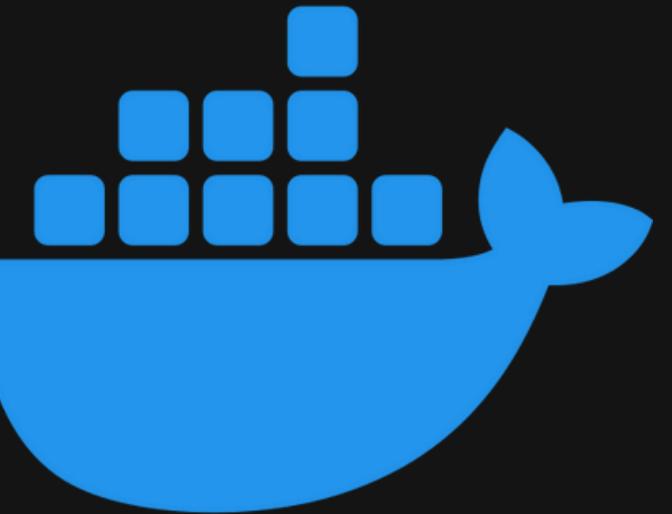


-VV(・)VV-

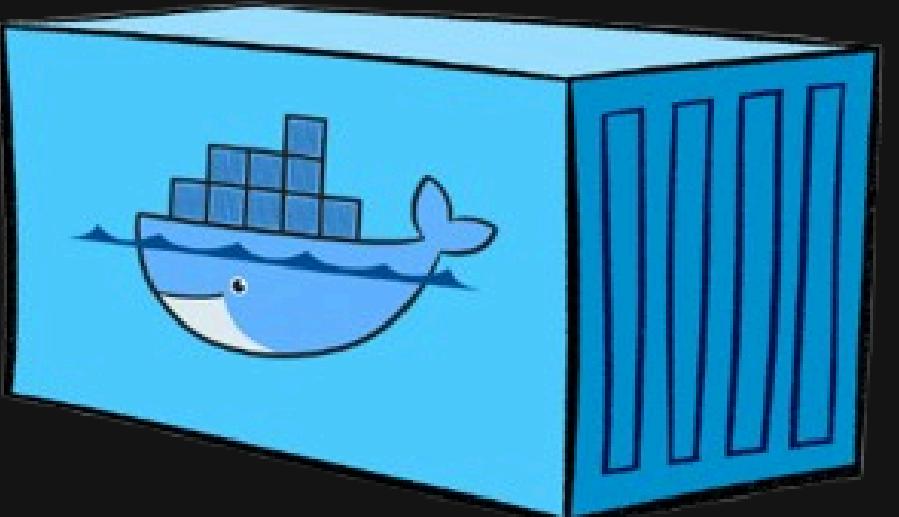
NA MINHA MÁQUINA  
**FUNCIONA!**

# INTRODUZINDO DOCKER

O Docker foi criado em 2013 por Solomon Hykes, junto com a equipe da empresa dotCloud (que depois mudou o nome para Docker Inc.).



Uma ferramenta open source de  
virtualização leve

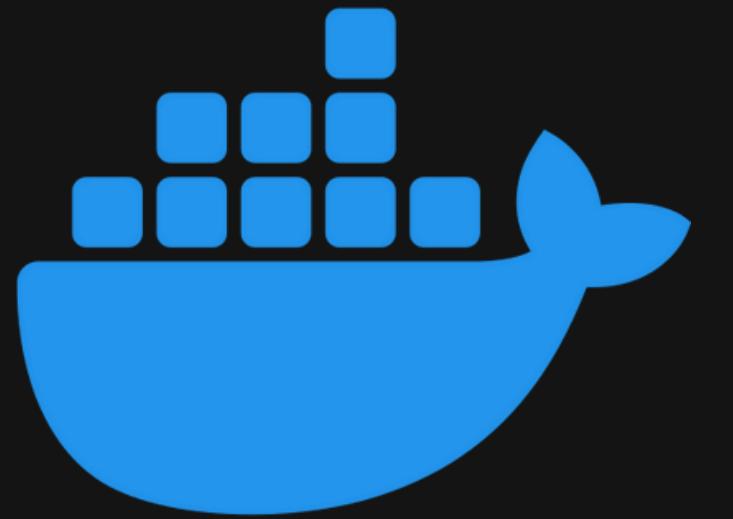
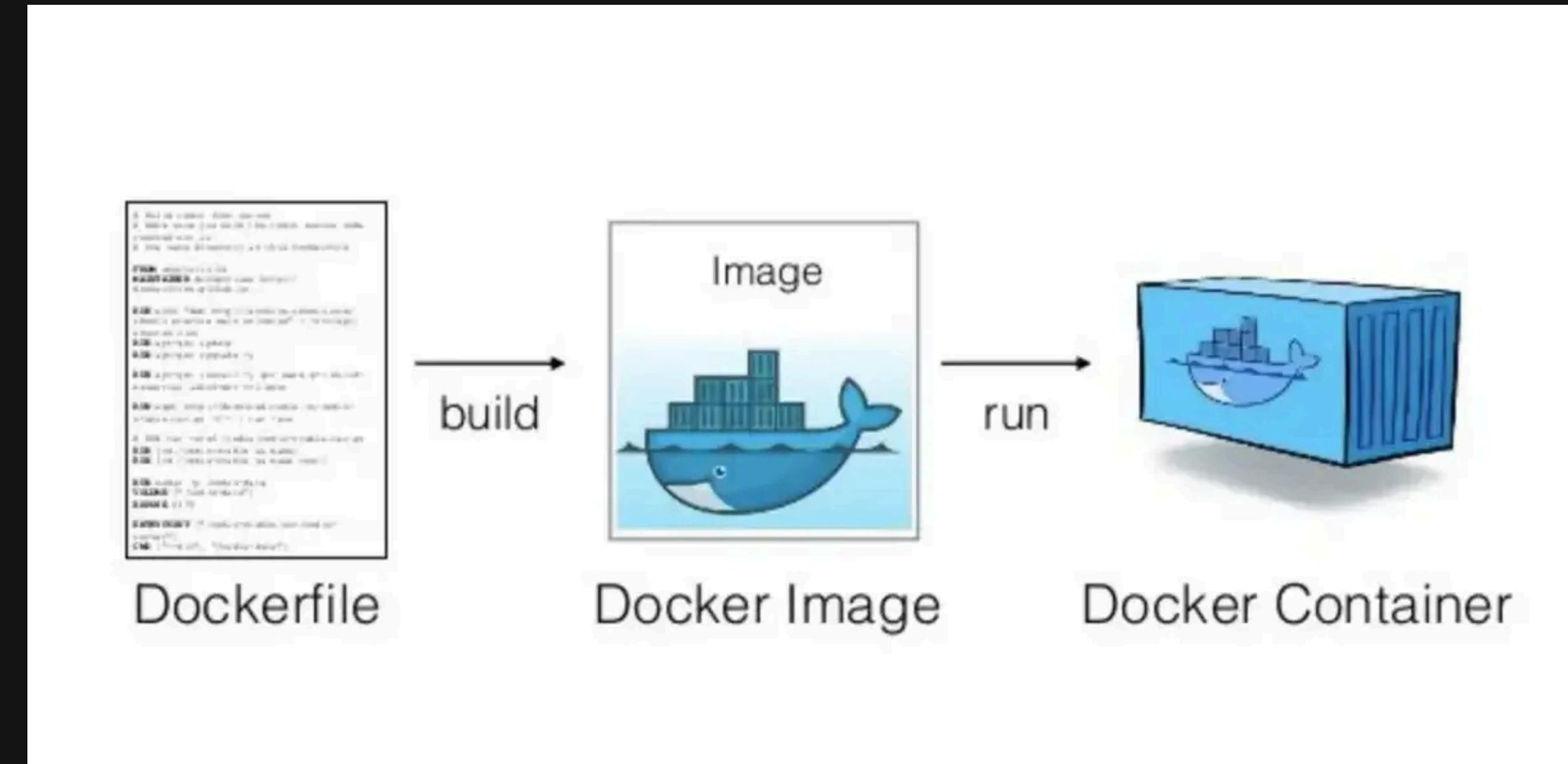


Roda aplicações isoladas em containers

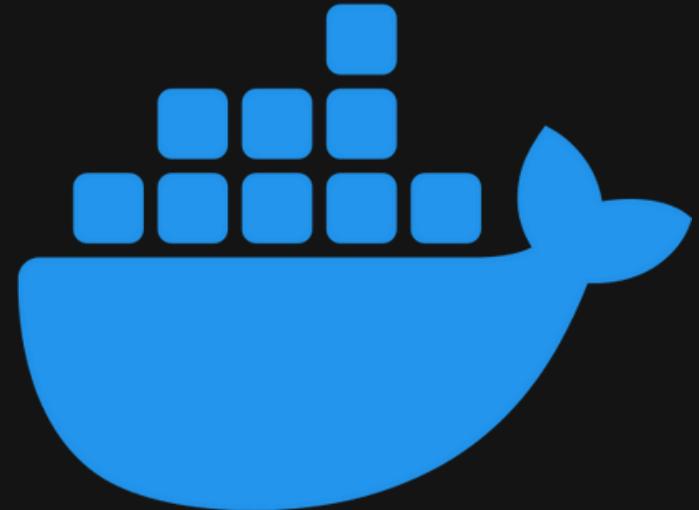


# PORQUE USAR DOCKER?

- Pacote completo → junta código, bibliotecas e configurações num só lugar.
- Segurança e organização → variáveis e senhas guardadas direitinho.
- Funciona em qualquer lugar → mesma versão em qualquer computador ou servidor.



# INSTALANDO DOCKER



Windows / macOS

- Baixar e instalar o Docker Desktop:
- <https://www.docker.com/products/docker-desktop>
- Seguir os passos do instalador.
- Reiniciar a máquina (se pedir).
- Verificar no terminal:



`docker --version`

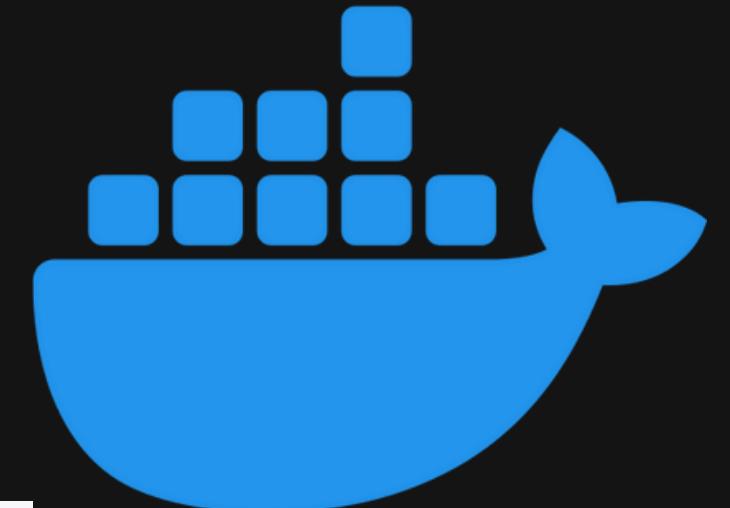


`wsl --install`

No Windows, o Docker Desktop usa o WSL2  
(Subsistema Linux).

Após instalar, já é possível rodar containers,  
criar imagens e usar o docker-compose.

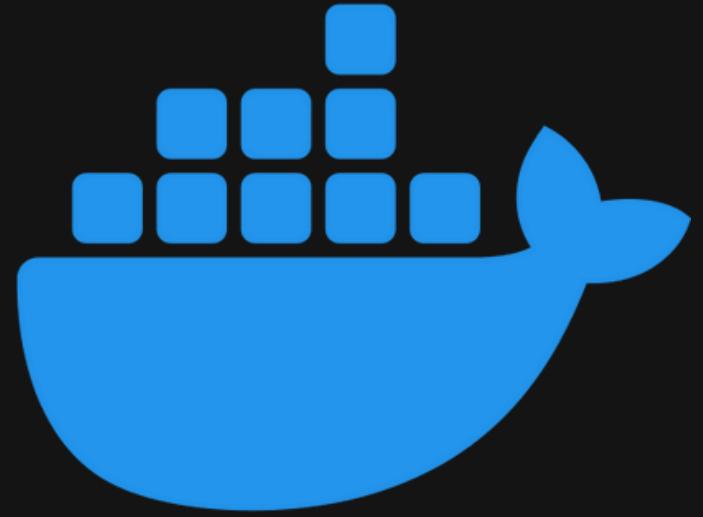
# INSTALANDO DOCKER



1. Clique em Sign Up
2. Escolha Continue with GitHub
3. Autorize o acesso e pronto!

The image shows the Docker sign-in page. At the top is the Docker logo. Below it is the text "Sign in" and a note: "Using Docker for work? We recommend signing in with your work email address." There is a text input field labeled "Username or email address\*" with a blue border. Below the input field is a blue "Continue" button. Underneath the button is the text "OR". Below "OR" are three other sign-in options: "Continue with Google" (with a Google "G" icon), "Continue with GitHub" (with a GitHub icon), and "Continue with SSO" (with a key icon). At the bottom of the page is a link: "Don't have an account? [Sign Up](#)".

# CONCEITOS BÁSICOS



**Container:** execução isolada do app.

**Imagen:** definição de como o container vai funcionar.

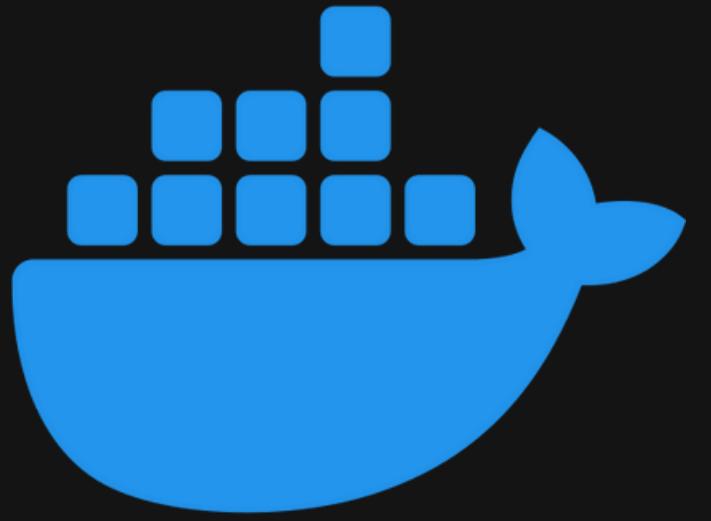
**Porta:** comunicação do container com o mundo externo.

**Volume:** pasta compartilhada, persiste dados mesmo que o container seja destruído.

**Registro / Registry:** armazena imagens (Docker Hub)

Detalhamento de cada conceito a seguir →

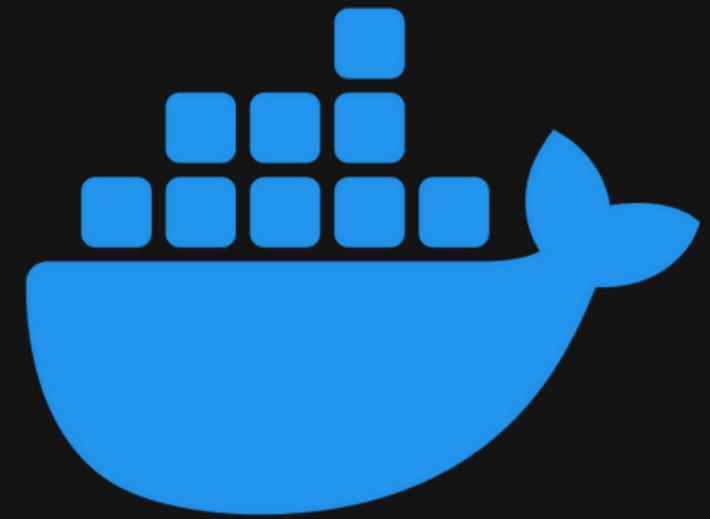
# CONTAINER



É como se fosse uma caixinha onde roda a aplicação. Dentro dele vai estar tudo que a aplicação precisa (bibliotecas, dependências, configs), separado do resto do sistema.

Exemplo: um container só do Postgres, outro só do Node.js.

# IMAGEM

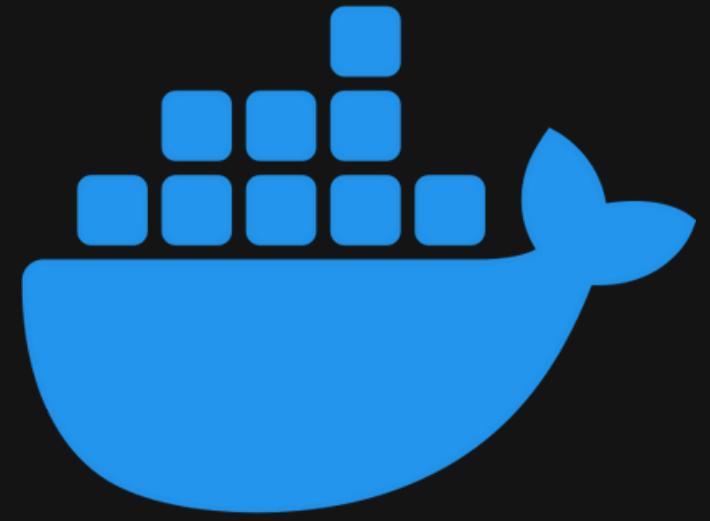


É o “molde” usado pra criar containers. Pensa como uma receita: a imagem diz como o container deve ser montado.

Exemplo: node:18-alpine é uma imagem oficial do Node mais leve.

# POR

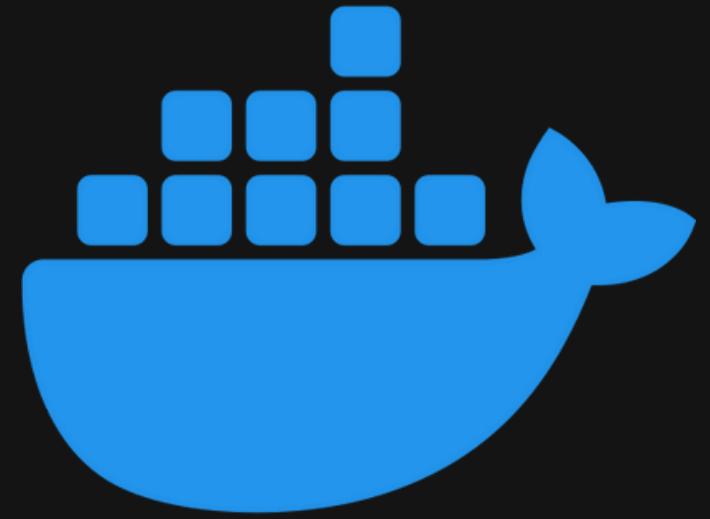
# T



É a forma de expor o container pro mundo externo. Como ele é isolado, precisa de uma “porta de entrada/saída” pra comunicação.

Exemplo: `-p 3000:3000` mapeia a porta do container pra máquina.

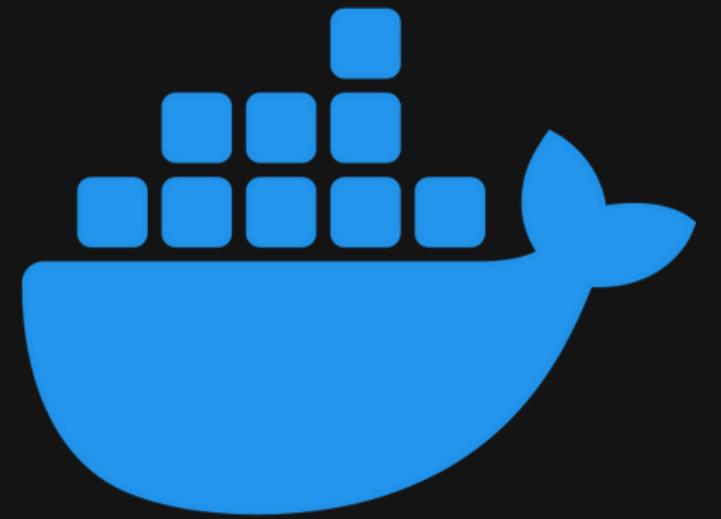
# VOLUME



Área de armazenamento compartilhada. Mesmo que o container morra, os dados ficam salvos.

Exemplo: banco de dados em Docker usa volume pra não perder as tabelas ao reiniciar.

# REGISTRO / REGISTRY



É onde ficam guardadas as imagens (tipo um repositório).

Exemplo: Docker Hub (público) ou um registry privado da empresa

# DIFERENÇA DE IMAGEM E CONTAINER

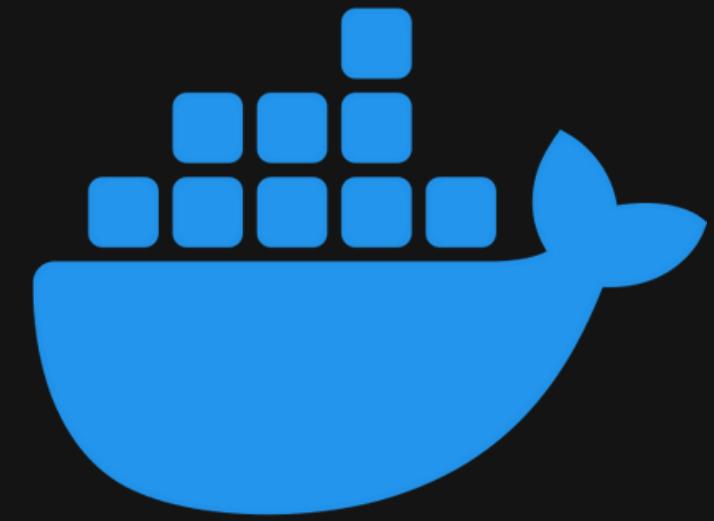
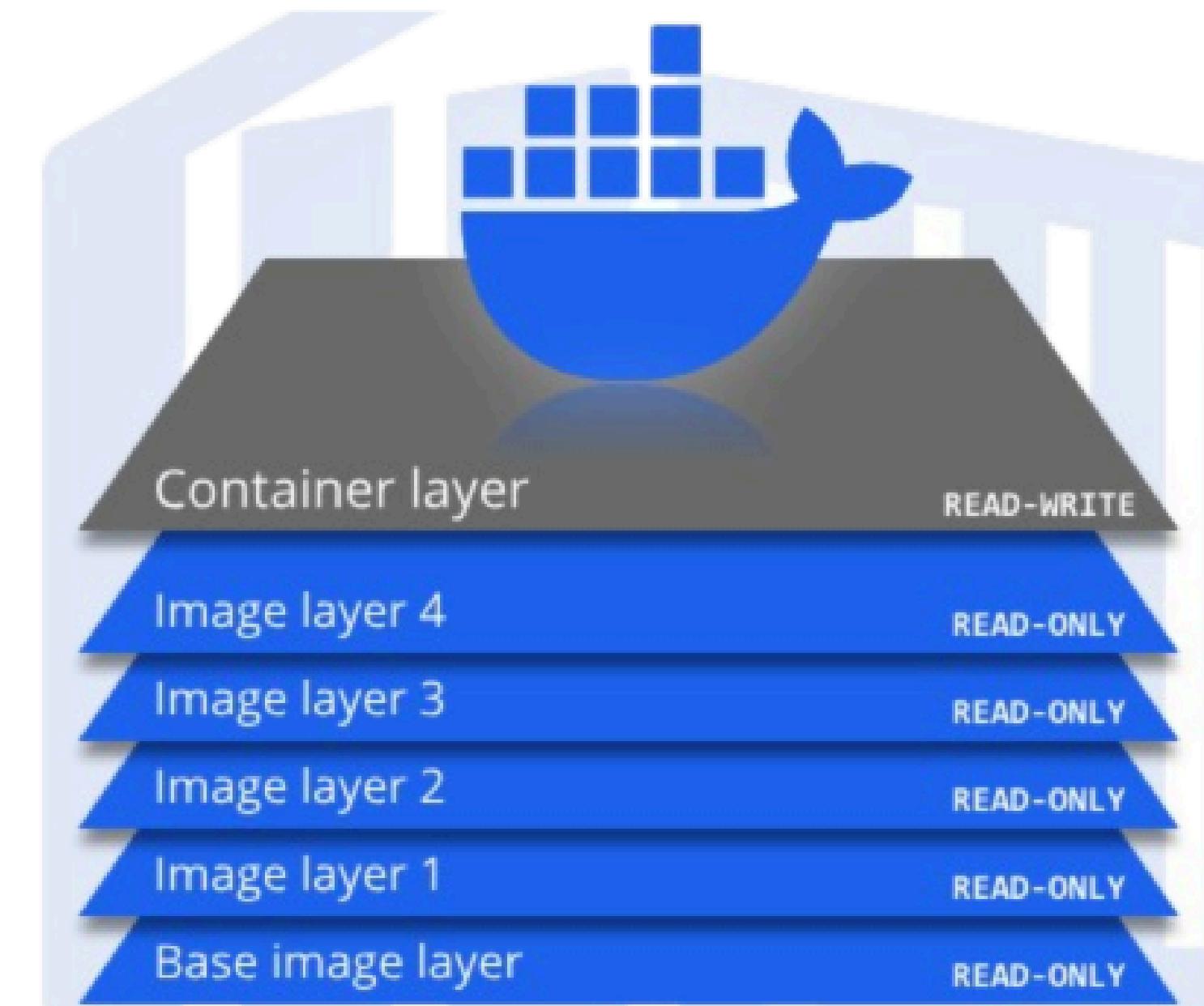


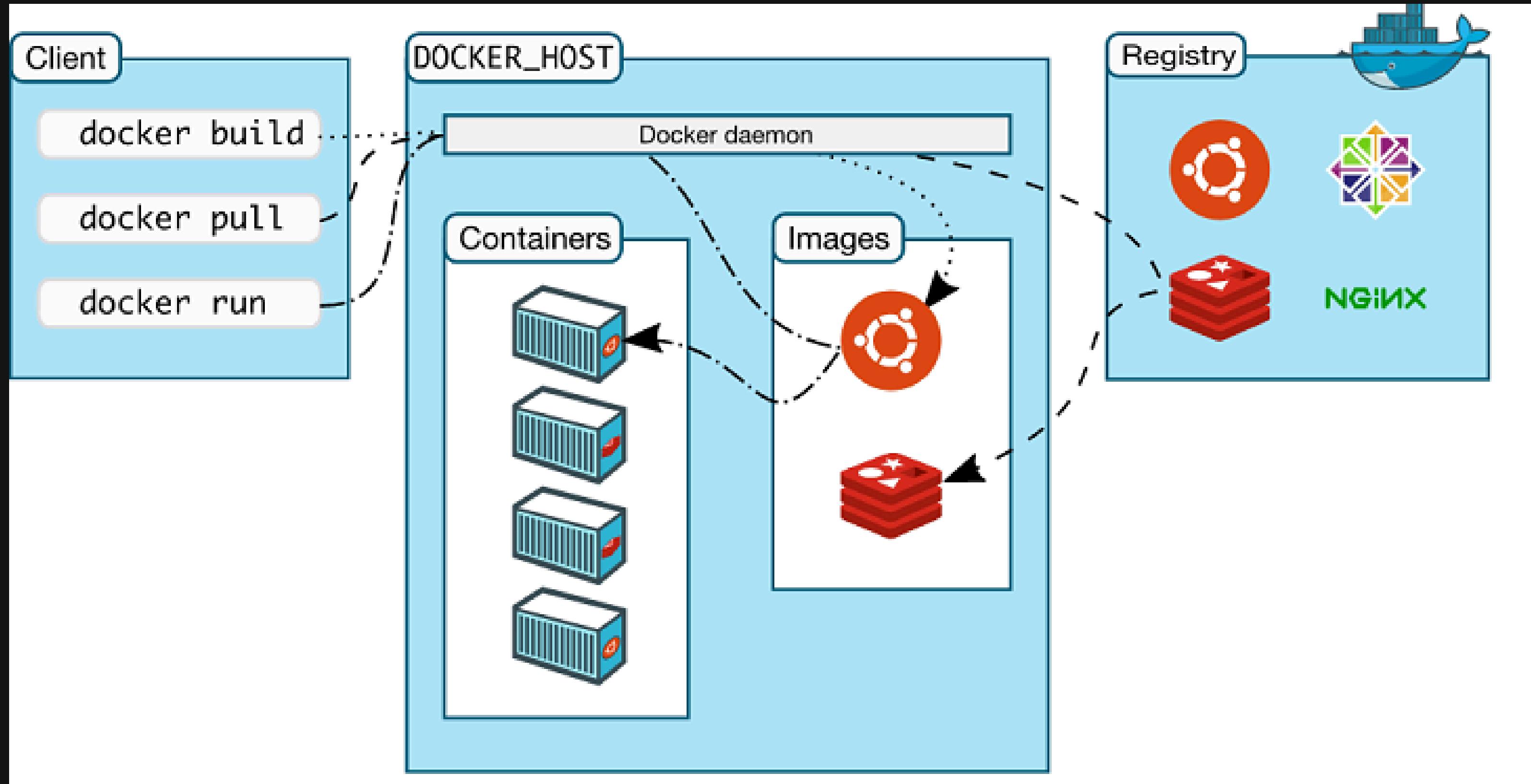
Imagen → camadas read-only com código + dependências

Container → adiciona camada read-write sobre a imagem

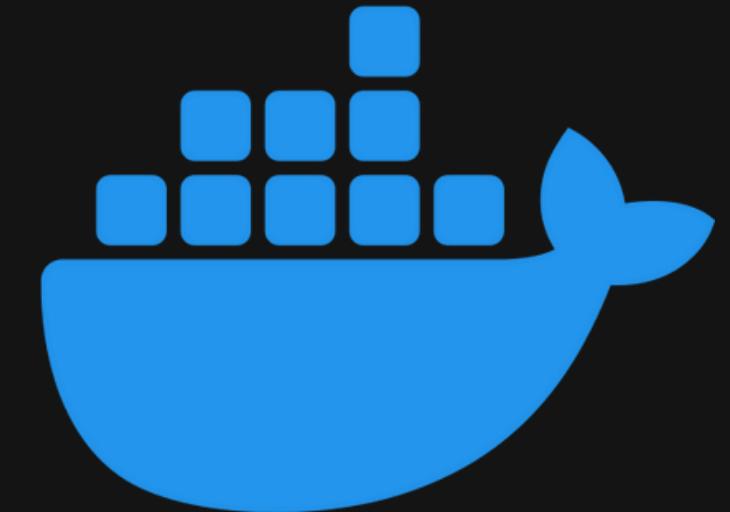
Vários containers podem ser criados da mesma imagem



# ARQUITETURA DO DOCKER



# COMANDOS BÁSICOS



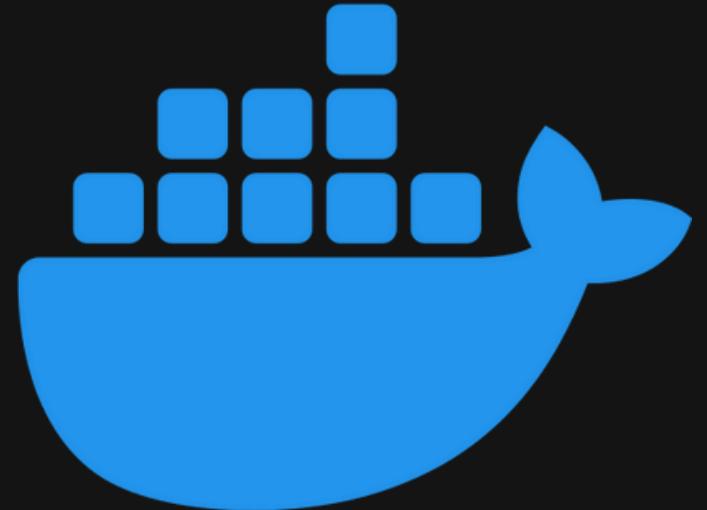
## Imagens



```
docker images  
docker image ls  
docker pull nginx  
docker build .
```

```
# listar todas as imagens  
# mesmo comando acima  
# baixar imagem do registry  
# criar imagem do Dockerfile atual
```

# COMANDOS BÁSICOS

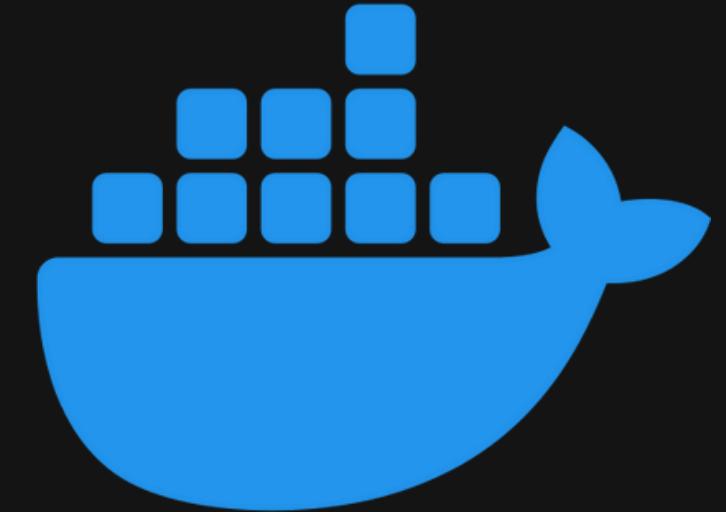


## Executando containers



```
docker run nginx          # executar container básico  
docker run -d nginx      # executar em background (detached)  
docker run -d -p 8080:80 nginx  # mapear porta host:container  
docker run -d -P nginx    # mapear porta aleatória
```

# COMANDOS BÁSICOS



<http://localhost:8080>

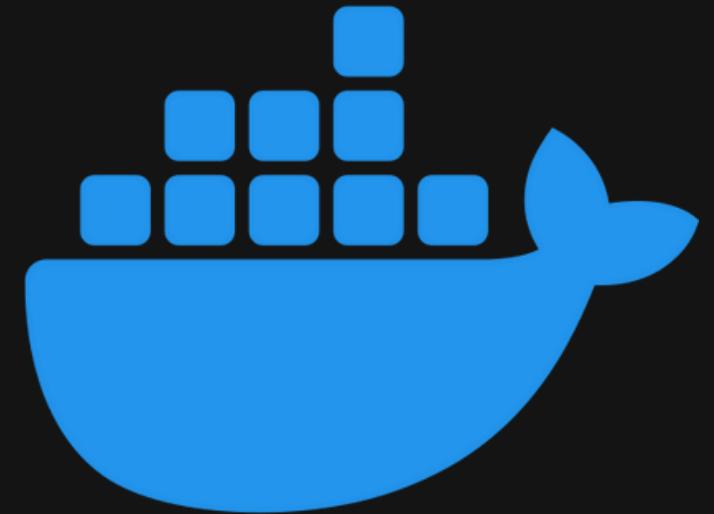
**Welcome to nginx!**

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to [nginx.org](http://nginx.org).  
Commercial support is available at [nginx.com](http://nginx.com).

*Thank you for using nginx.*

# COMANDOS BÁSICOS



```
# Executa um comando dentro do container  
docker exec -it <id_container> <comando>
```

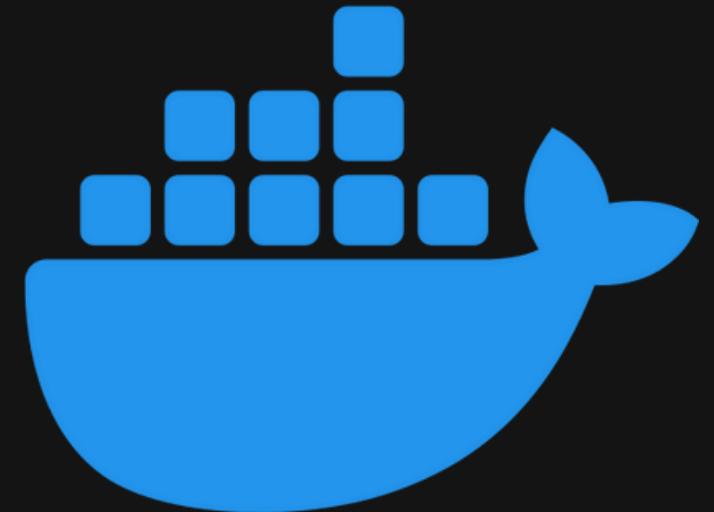
```
# Pausa e volta o container  
docker pause <id_container>  
docker unpause <id_container>
```

```
# Remove um container parado  
docker rm <id_container>
```

```
# Força a remoção de um container ativo  
docker rm -f <id_container>
```

```
# Remove todos os containers parados  
docker container prune
```

# COMANDOS BÁSICOS



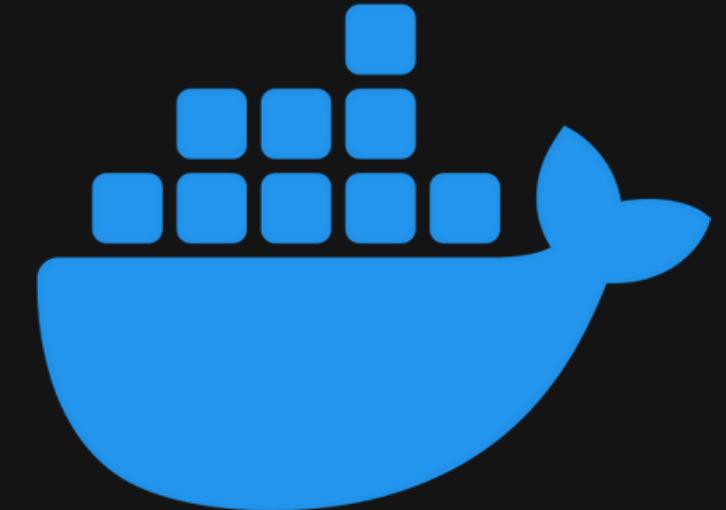
## Monitorar Containers



```
docker ps  
docker ps -a  
docker inspect <id>  
docker stats <id>  
docker attach <id>
```

```
# containers rodando  
# todos os containers (rodando + parados)  
# informações detalhadas do container  
# uso de CPU, memória, rede em tempo real  
# conectar ao stdout do container
```

# EXEMPLO PRÁTICO



## Consumindo a imagem do nginx

```
# 1. Executar nginx em background na porta 8080
docker run -d -p 8080:80 nginx

# 2. Testar se está funcionando
curl localhost:8080                                # retorna HTML do nginx

# 3. Acessar pelo navegador
# http://localhost:8080 ou http://SEU_IP:8080

# 4. Ver containers rodando
docker ps

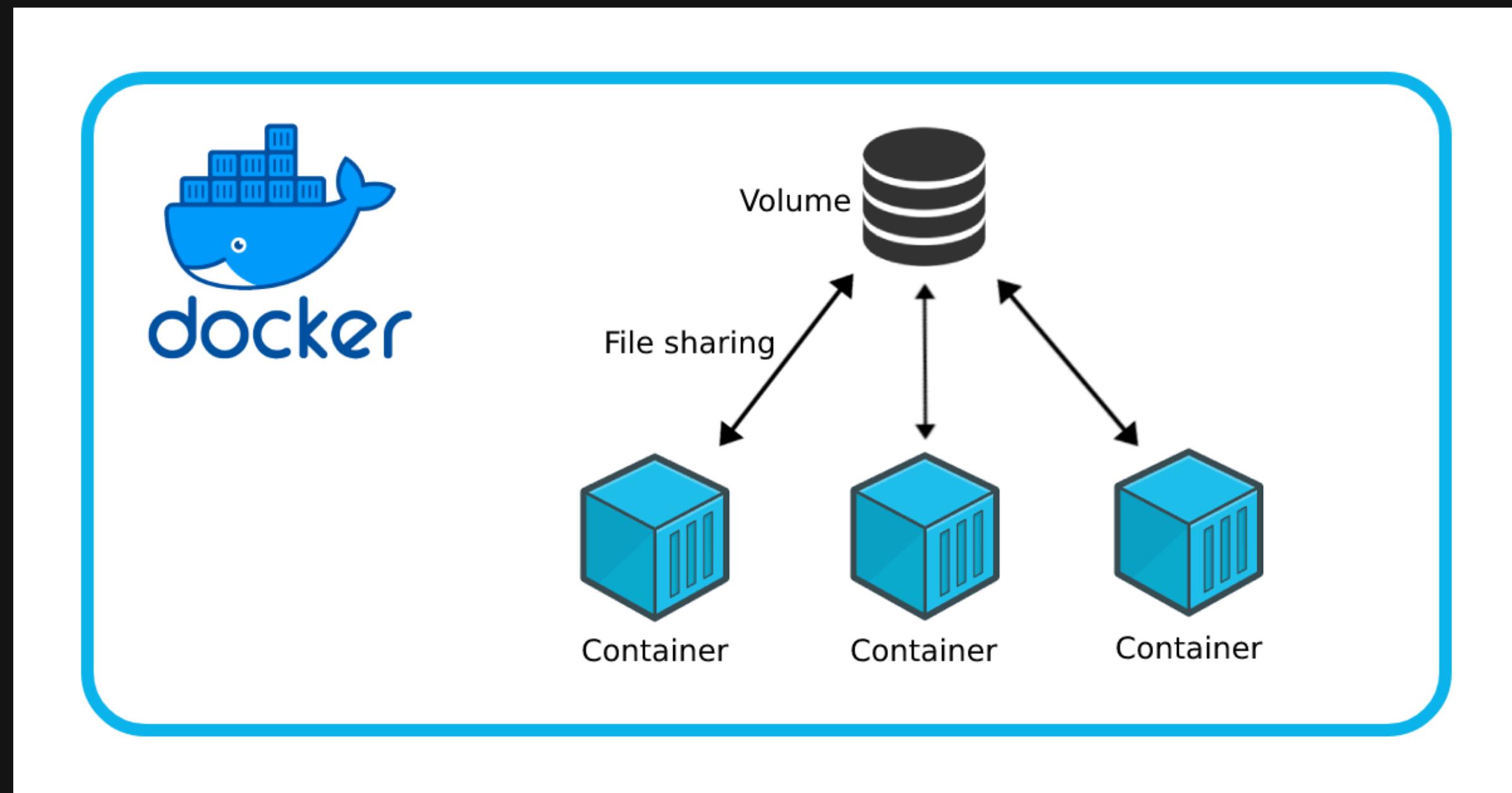
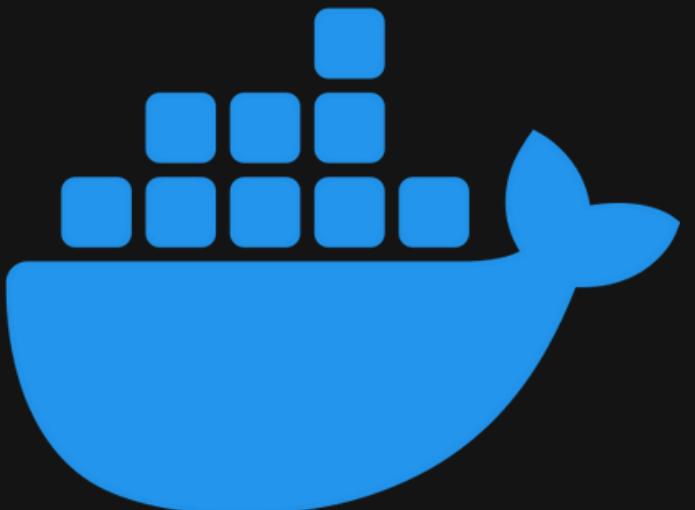
# 5. Parar o container
docker stop <container_id>
```

# VOLUMES NO DOCKER

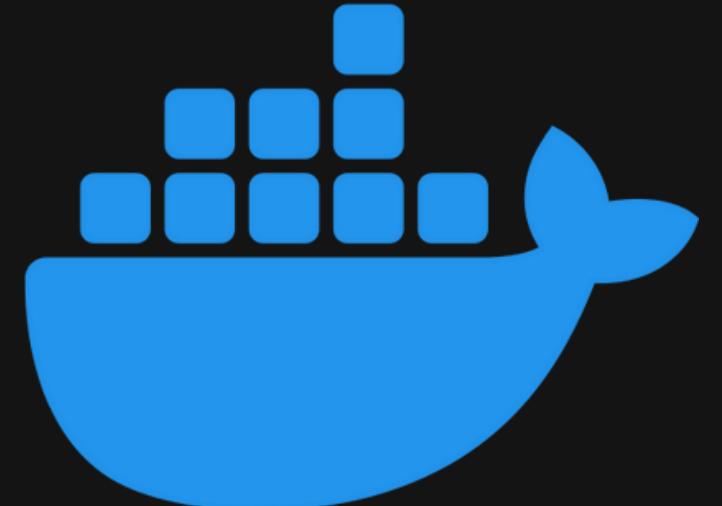
O que são volumes no docker?

É um espaço de armazenamento **fora da camada de escrita** do container.

Dados **não se perdem** se o container for removido.



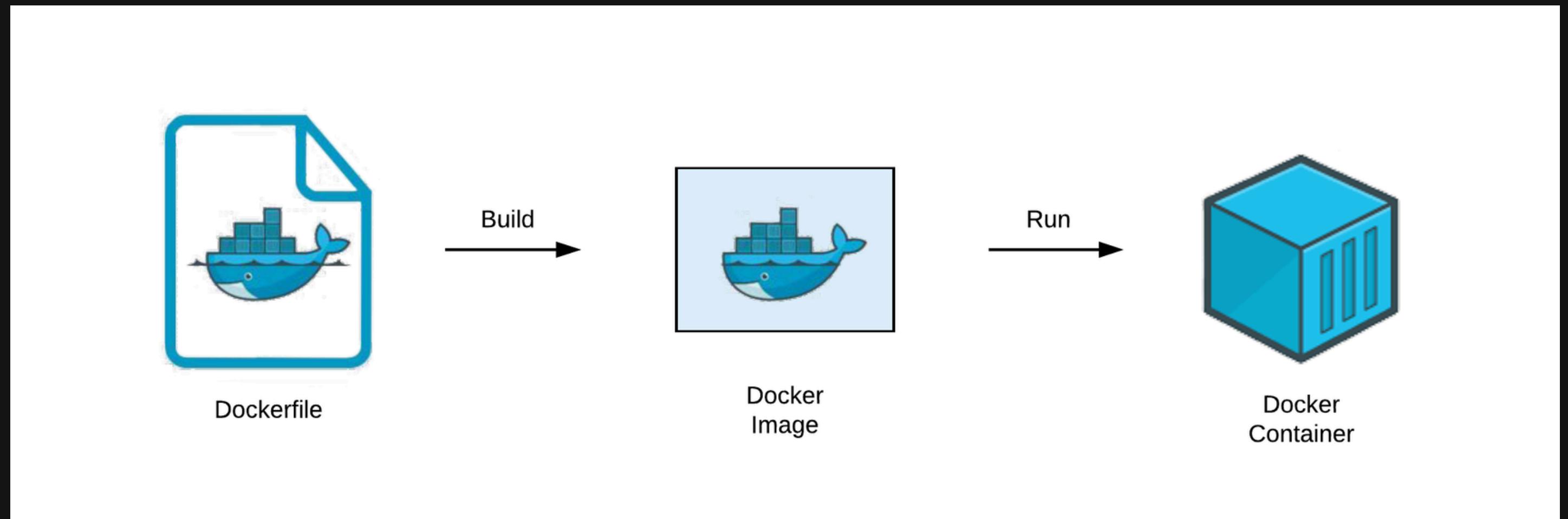
# O QUE É UMA DOCKERFILE?



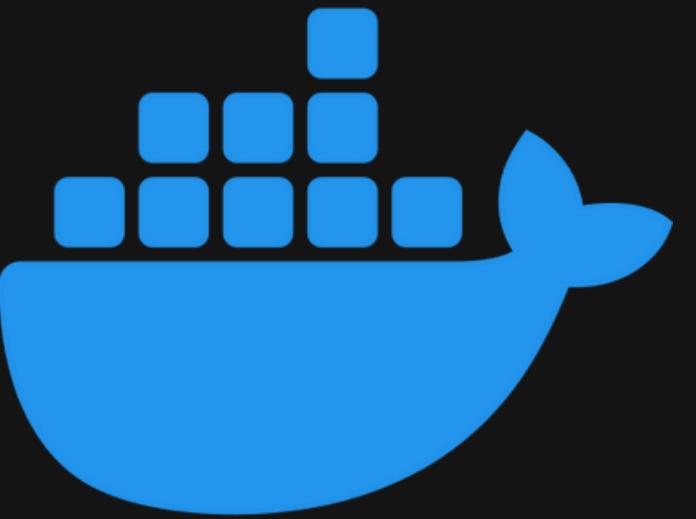
É uma sequência de comandos que vai ditar do que a minha aplicação precisa.

Essa sequência de passos é definida da seguinte forma:

- FROM
- RUN
- COPY
- ENV
- WORKDIR
- EXPOSE
- CMD



# FROM

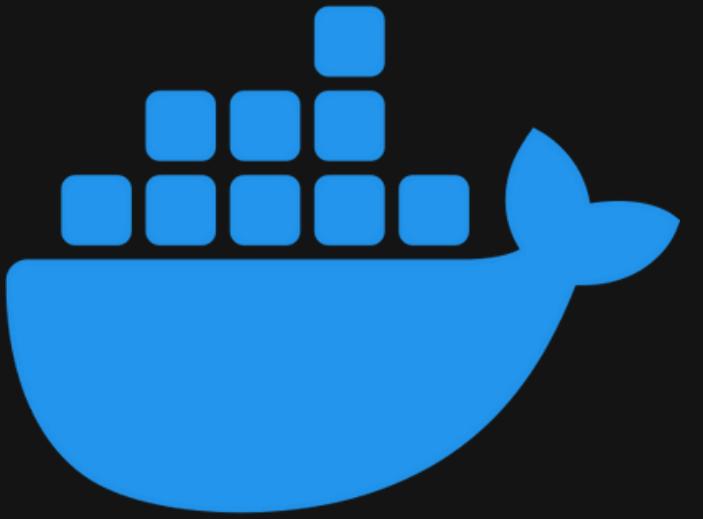


É o ponto de partida. Tu diz de qual imagem base vai herdar.



FROM node:18

# RUN

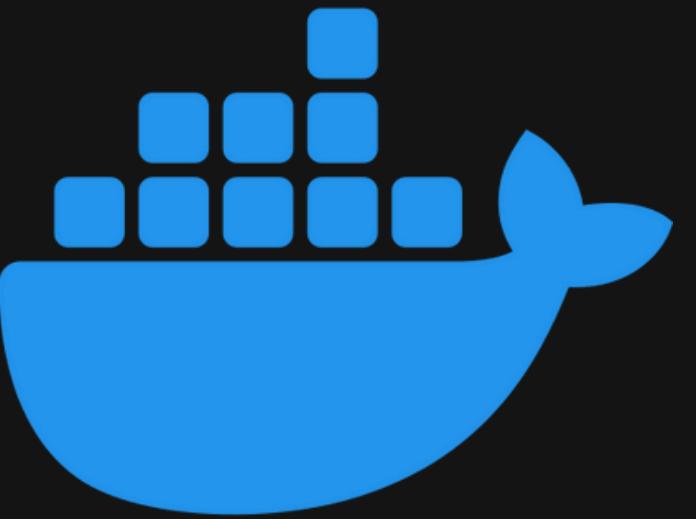


Executa comandos durante a construção da imagem.



**RUN** npm install

# COPY

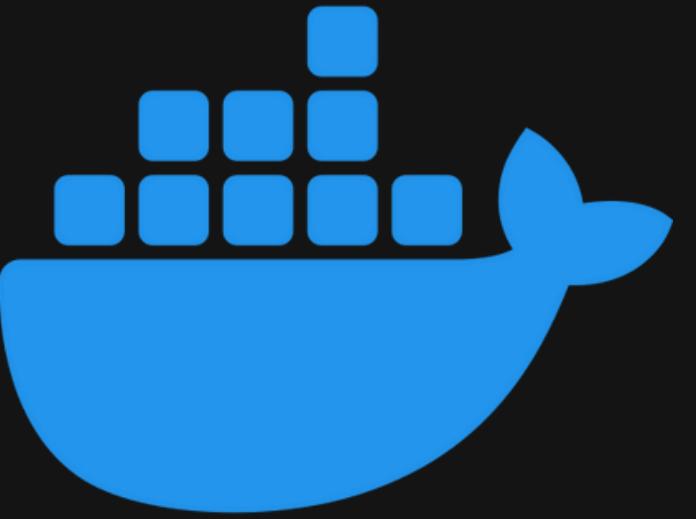


Copia arquivos do computador para dentro do container.



COPY . . .

# ENV

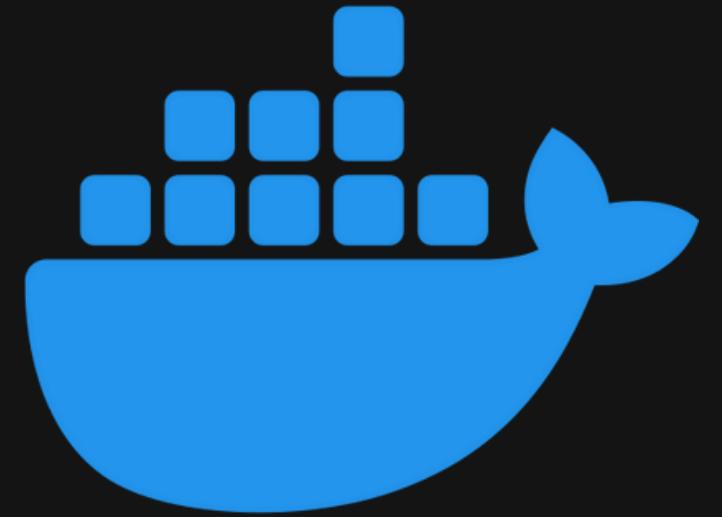


Define variáveis de ambiente dentro do container.



```
ENV NODE_ENV=production
```

# WORKDIR

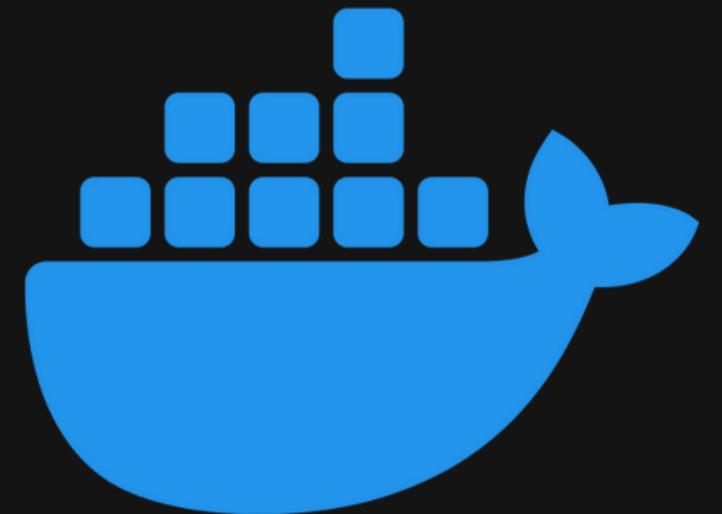


Define qual pasta será o "diretório atual" dentro do container.



WORKDIR /app

# EXPOSE

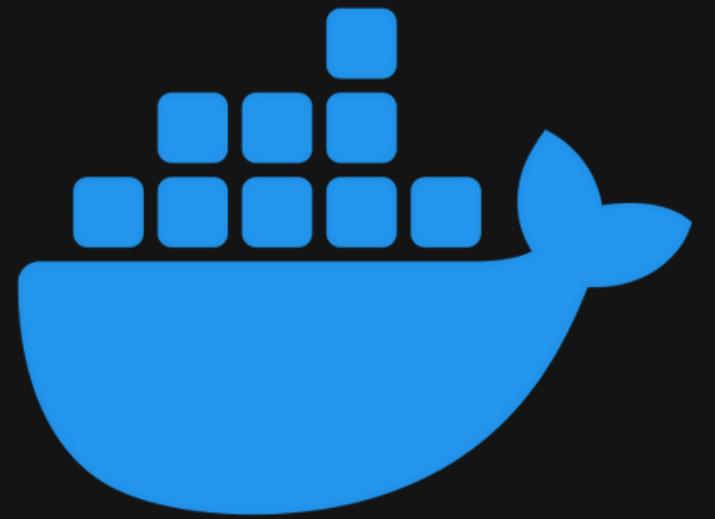


Informa que a aplicação vai usar a porta a ser definida internamente.



EXPOSE 3000

# CMD

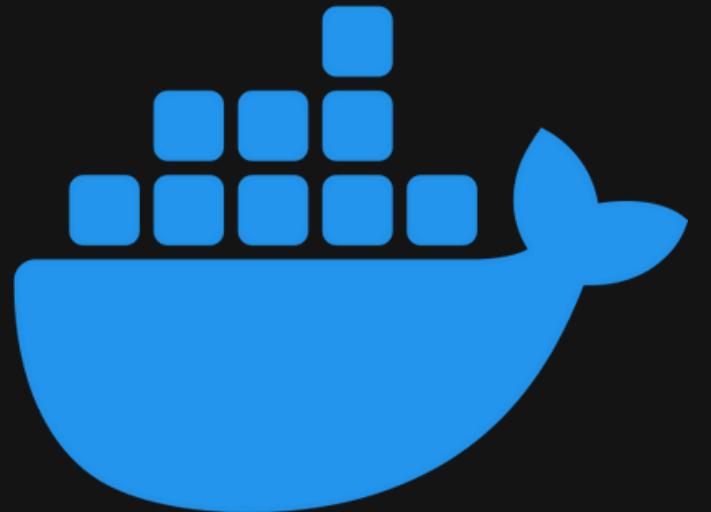


Comando final que roda quando o container é iniciado.



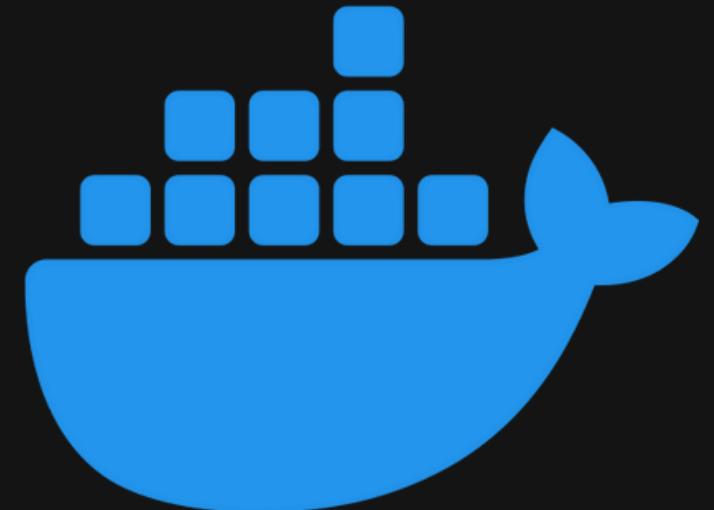
CMD ["npm", "start"]

# CACHE EM DOCKER



- Cada comando que eu executo é uma nova camada.
- temos uma hierarquia entre as camadas, ou seja, a ordem importa.
- se eu modificar o diretório do projeto, o comando **COPY** copia todo o projeto novamente porque houve alteração.
- poderíamos adaptar o build apenas para o código principal, ou seja, fazendo um **COPY** apenas para o código em si.
- ao invés de cada **COPY** que for executado

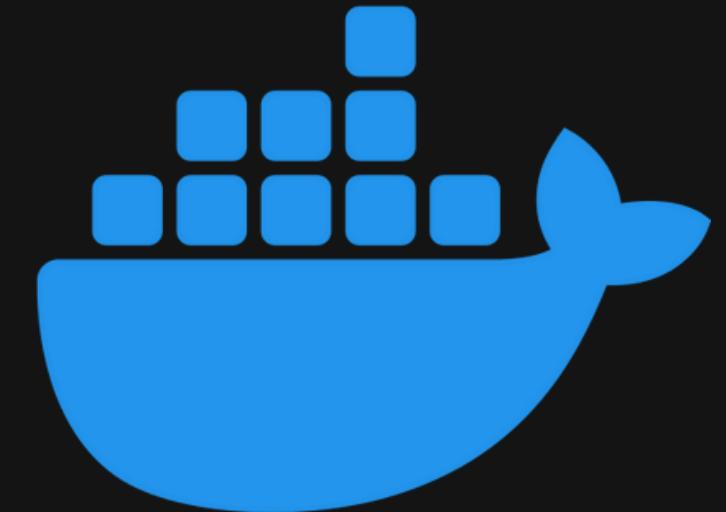
# EXEMPLO DE UMA DOCKERFILE



Exemplo ao lado  
mostra uma Dockerfile  
para um app Node.js.

```
FROM node:18-alpine
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
EXPOSE 3000
CMD ["npm", "start"]
```

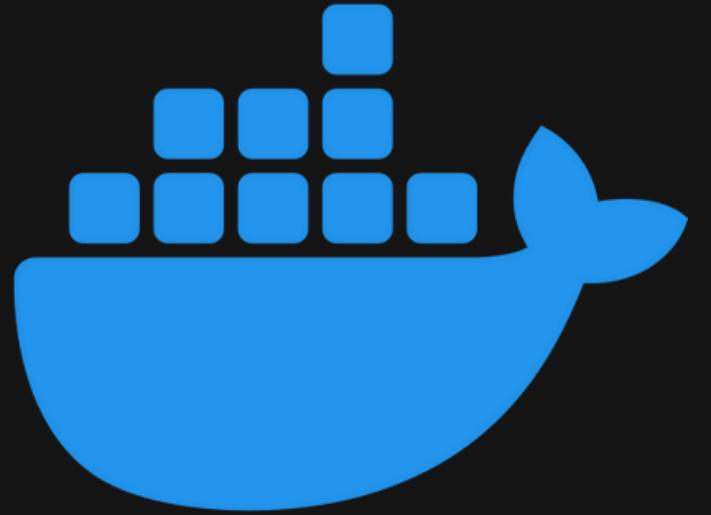
# RODANDO O CONTAINER



```
# 1 Construir a imagem e definir o nome  
docker build -t meu-app .
```

```
# 2 Rodar um container a partir da imagem  
docker run -d -p 3000:3000 meu-app
```

# BOAS PRÁTICAS EM DOCKER



- Usar imagens leves (ex.: node:18-alpine).
- Minimizar camadas no Dockerfile.
- Usar .dockerignore (mesmo conceito do .gitignore).
- Sempre versionar o Dockerfile junto com o código.