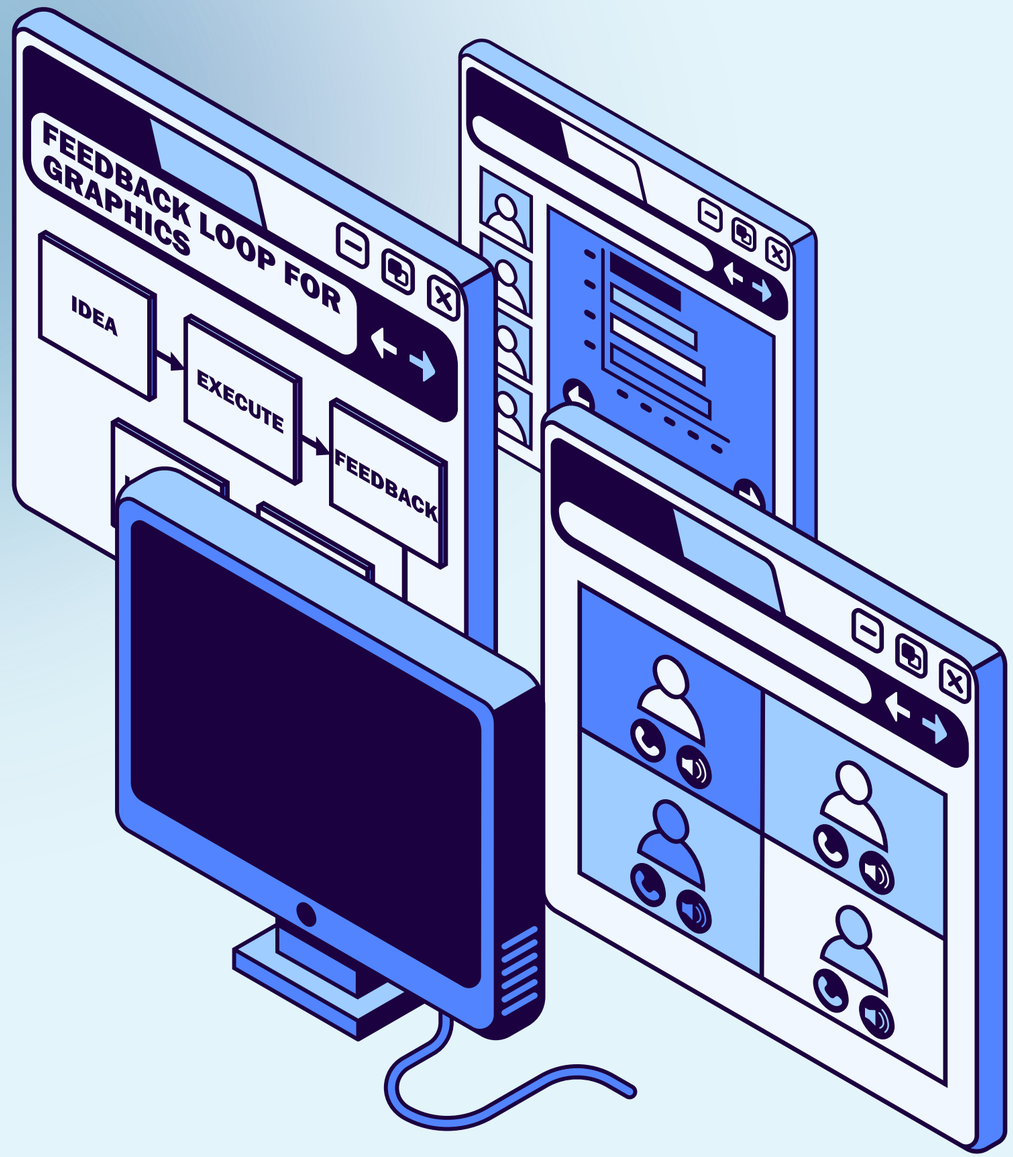
DeployPRO

Do código ao deploy

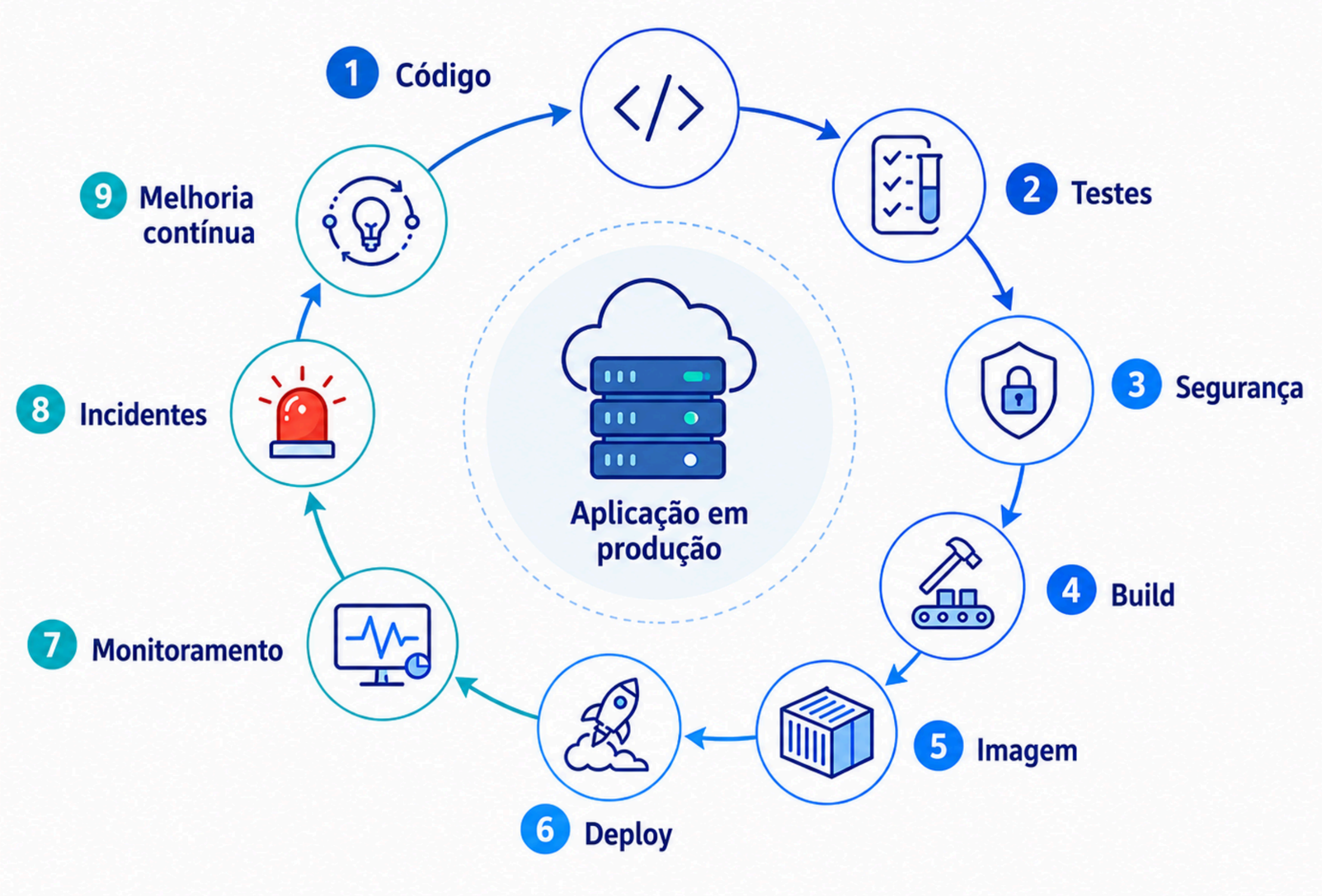
Aprenda a publicar uma aplicação real usando Docker, GitHub Actions, AWS e práticas de produção



Módulo 1 – Mentalidade DevOps e visão de produção

- O ciclo real de uma app
- Diferença entre desenvolvimento, infraestrutura, cloud, SRE e plataforma.
- Por que “funcionar na minha máquina” não é suficiente.
- Principais dores em produção
- O que é devops de verdade

O ciclo real de uma aplicação



O ciclo real de uma aplicação

1 - Código
Tudo começa no código-fonte: funcionalidades, correções, configurações e dependências da aplicação.

2 - Testes
Antes de seguir para produção, o código precisa ser validado para reduzir erros, quebrar menos funcionalidades e aumentar a confiança na entrega.

3 - Segurança
A aplicação e suas dependências devem passar por verificações de vulnerabilidades, boas práticas e exposição indevida de dados ou credenciais.

4 - Build
O código é preparado para execução. Aqui entram instalação de dependências, compilação, empacotamento e validações básicas.

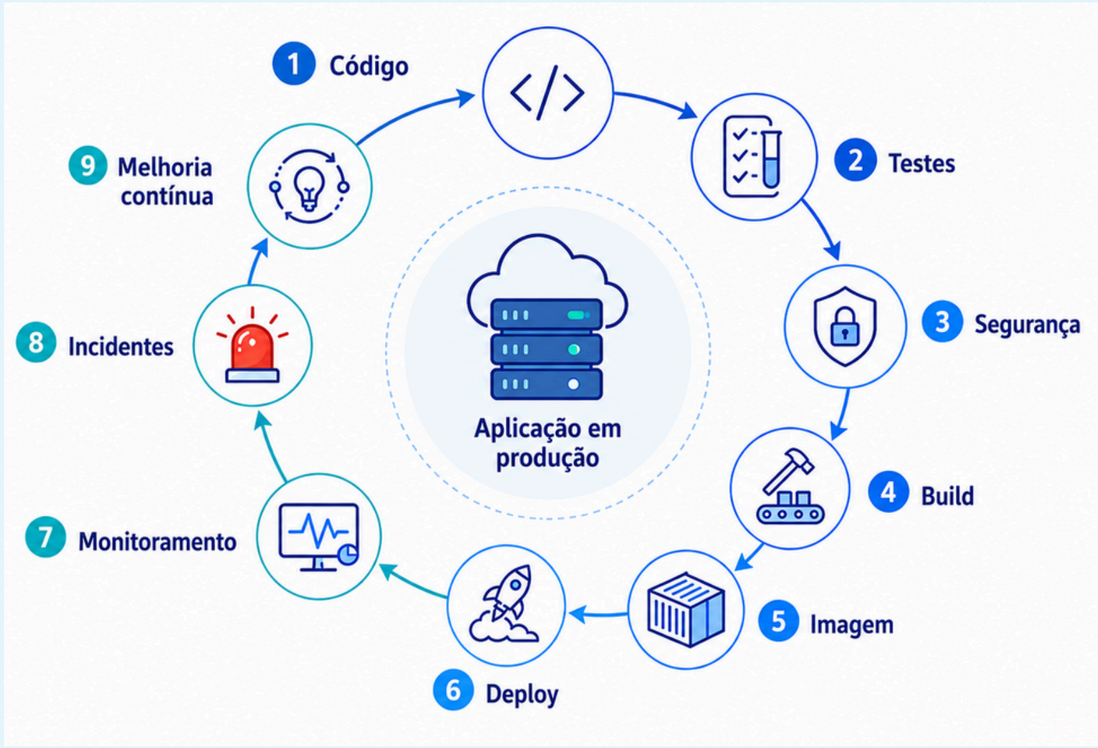
5 -Imagem / Artefato
A aplicação é transformada em uma imagem de container, com tudo que precisa para rodar de forma padronizada em qualquer ambiente.

6 - Deploy
A imagem é publicada em um ambiente real, como servidor, cloud ou cluster, ficando disponível para os usuários.

7 - Monitoramento
Métricas, disponibilidade, consumo de recursos e performance são acompanhados para entender a saúde da aplicação.

8 - Incidentes
Quando algo falha, o time precisa identificar o problema, corrigir rapidamente e reduzir o impacto para o usuário.

9 - Melhoria contínua
Cada erro, métrica ou feedback gera aprendizado para melhorar o código.



Diferença entre desenvolvimento, infraestrutura, cloud, SRE e plataforma.

🛠️ Desenvolvimento (Dev)

- **O que faz**: Cria as aplicações, funcionalidades e regras de negócio.
- **Foco:** Entregar valor direto ao usuário final escrevendo código.
- **Metáfora:** São os arquitetos e pedreiros que constroem as casas e prédios da cidade.

🔌 Infraestrutura (Tradicional)

- **O que faz:** Gerencia o hardware físico, cabos, redes e servidores locais (on-premises).
- **Foco:** Manter os equipamentos físicos operacionais, seguros e conectados.
- **Metáfora:** É a empresa de energia e saneamento que enterra os canos e monta os postes físicos.

☁️ Cloud (Computação em Nuvem)

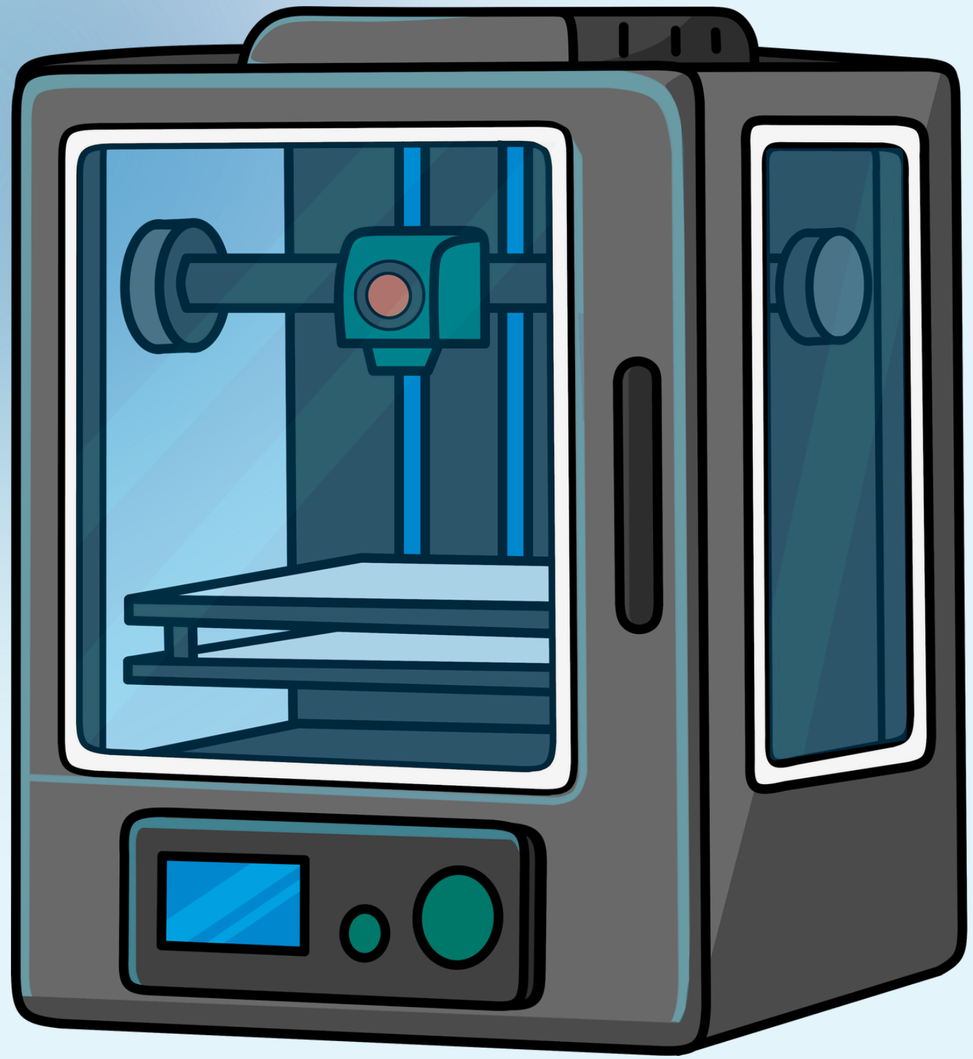
- **O que faz:** Gerencia a infraestrutura virtualizada em provedores como AWS, Google Cloud ou Microsoft Azure.
- **Foco:** Disponibilizar servidores, bancos de dados e redes de forma rápida por meio de software.
- **Metáfora:** É o fornecedor que aluga terrenos prontos e modulares no céu, dispensando a compra do chão físico.

📐 SRE (Site Reliability Engineering)

- **O que faz:** Aplica conceitos de engenharia de software para resolver problemas de operações.
- **Foco:** Confiabilidade, alta disponibilidade, monitoramento, performance e automação de incidentes.
- **Metáfora:** É a equipe de engenheiros de tráfego e defesa civil que monitora a cidade em tempo real para evitar blefautes e acidentes.

📦 Plataforma (Platform Engineering)

- **O que faz:** Cria portais de autoatendimento, ferramentas e esteiras automáticas de entrega para os desenvolvedores internos.
- **Foco:** Reduzir a carga cognitiva do time de desenvolvimento e acelerar os deploys.
- **Metáfora:** É a equipe que pavimenta as estradas e cria placas de sinalização padronizadas para que os construtores andem mais rápido.



DeployPRO

Módulo 2 - Linux, terminal e fundamentos de rede para deploy

Objetivo

Nivelar a turma nos fundamentos mínimos para não travar no restante do curso.

- terminal
- processos e portas
- env e permissões
- HTTP, DNS e IP
- Firewall e ssh
- Logs
- Prática

Linux, terminal e fundamentos

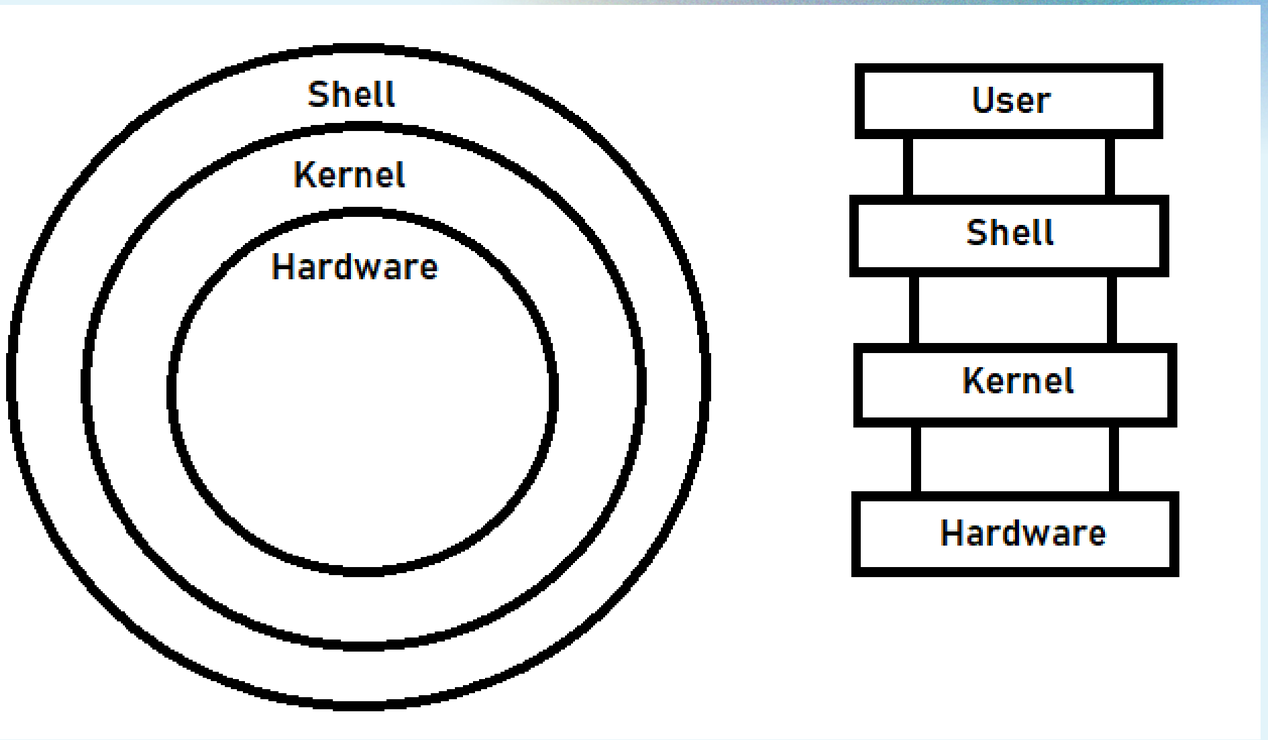
O Terminal / shell

No Linux, o shell é um programa que atua como uma interface de comunicação entre você e o sistema operacional. Ele recebe os comandos digitados em texto, os traduz para uma linguagem que o sistema entende (o kernel) e exibe os resultados na tela.

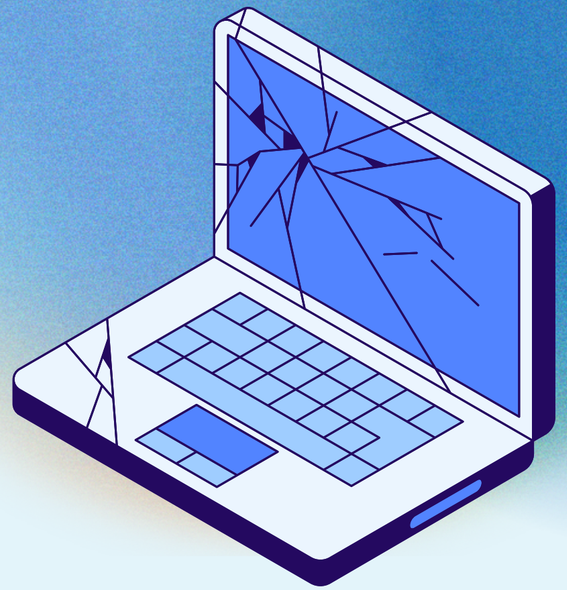
Linux, terminal e fundamentos

O Terminal / shell

No Linux, o shell é um programa que atua como uma interface de comunicação entre você e o sistema operacional. Ele recebe os comandos digitados em texto, os traduz para uma linguagem que o sistema entende (o kernel) e exibe os resultados na tela.



Linux, terminal e fundamentos



Navegar

$ pwd - onde estou?
$ ls - o que existe aqui?
$ cd - para onde vou?

Ler e buscar

$ cat - ler arquivo
$ grep - procura texto
$ tail - acompanha final do log

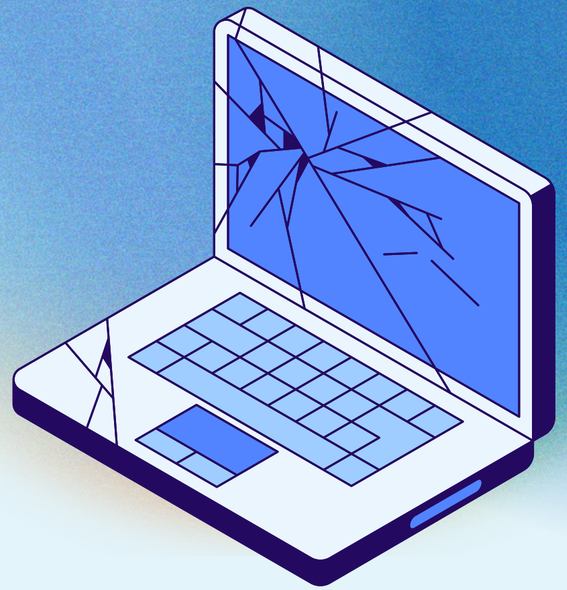
Executar

$ curl - testar HTTP
$ ps - ver processos
$ chmod - ajustes de permissão

Conectar

$ ssh - acessar servidor
$ scp - copiar arquivo remotamente
$ env - ver variaveis

Linux, terminal e fundamentos



Processos

O processo é um programa em execução.
Pode estar saudável, travado ou reiniciando.
$ ps aux | grep app
$ ss -lntp

Porta

Porta é o canal onde o processo aceita conexão: 80, 443, 3000, 5432, 3306

Variaveis de ambiente

Configurações externas ao código: porta, banco, credenciais e segredos.
Exemplo: PORT=3000, APP_ENV=prod
DATABASE_URL=db

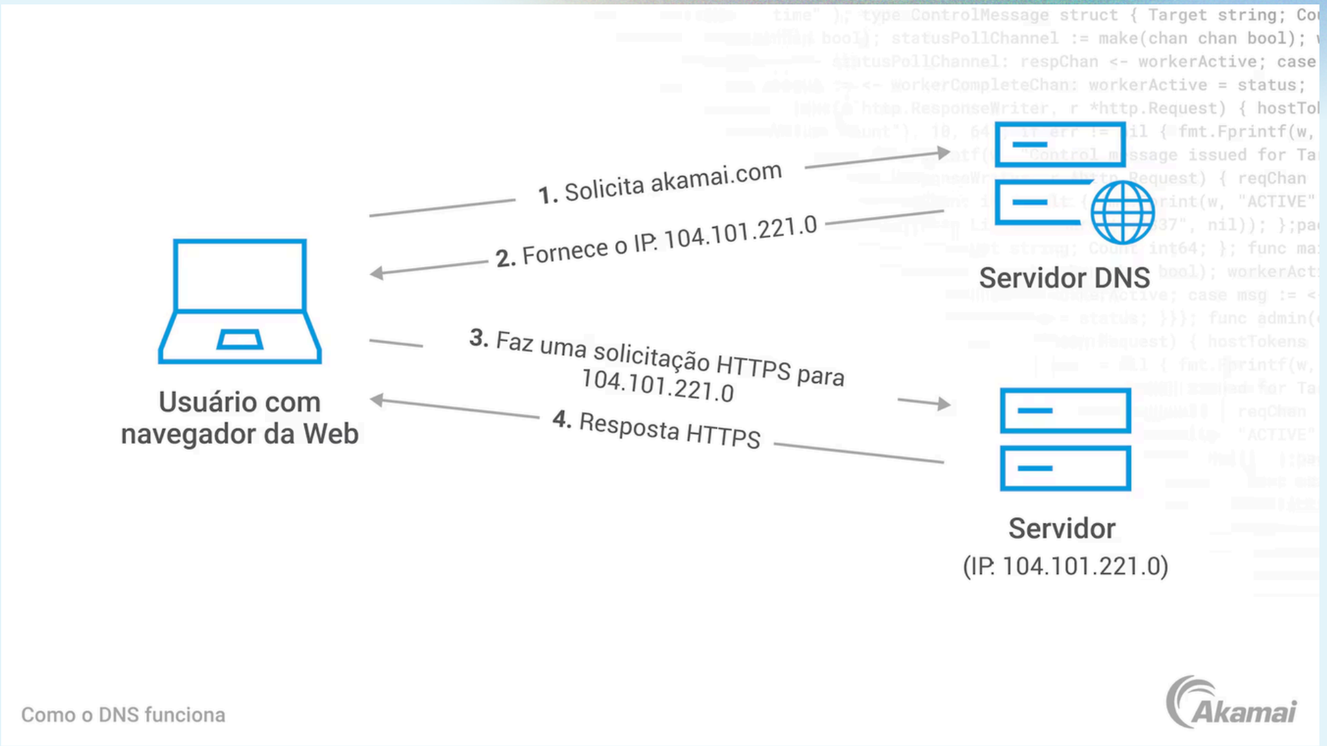
Para setar variáveis:
$ export NOME=valor

Para verificar variáveis de ambiente:
$ echo $VARIAVEL
$ env | grep VARIAVEL

Linux, terminal e fundamentos

DNS

O DNS (Sistema de Nomes de Domínio) é a "agenda telefônica" da internet. Ele converte nomes de sites fáceis de ler (como google.com) nos endereços IP numéricos que os computadores usam para se conectar (como 142.250.190.46).

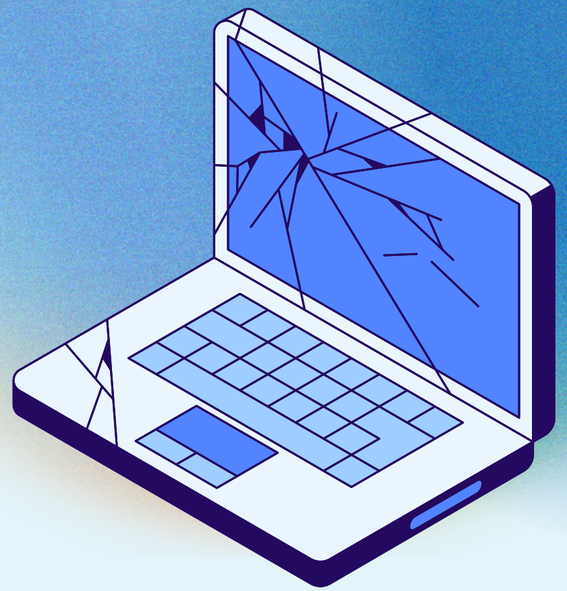


Linux, terminal e fundamentos

O que é IP?

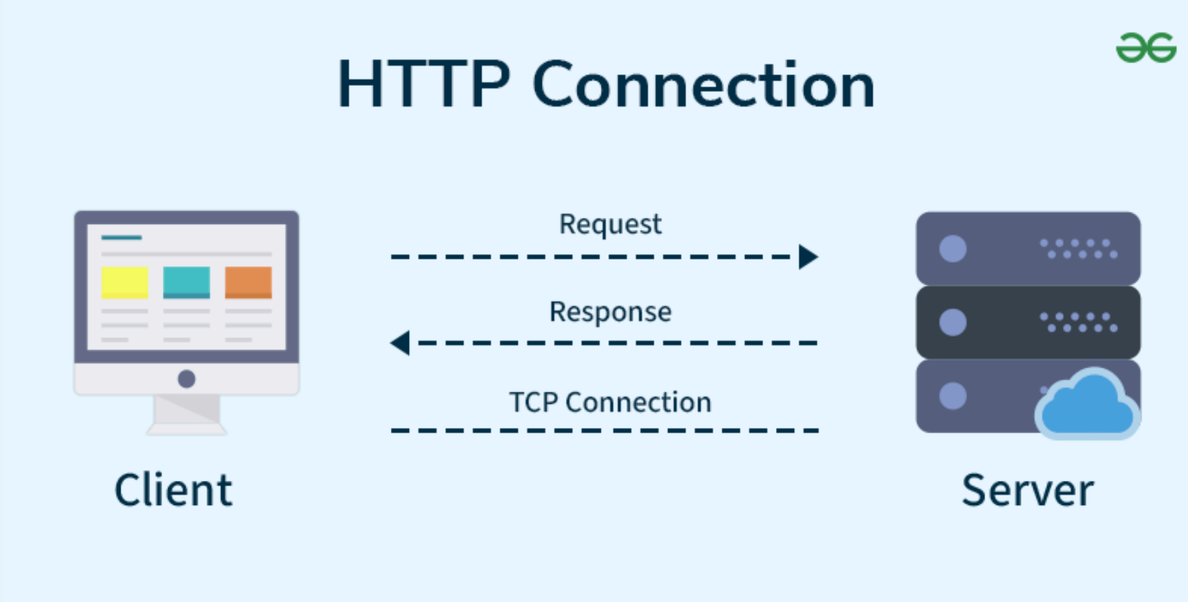
Refere a um **Endereço de Protocolo de Internet**. É um número de identificação exclusivo atribuído a qualquer dispositivo (como seu celular, computador ou impressora) para que ele possa se comunicar e ser localizado em uma rede. Pode ser público ou privado.

Linux, terminal e fundamentos



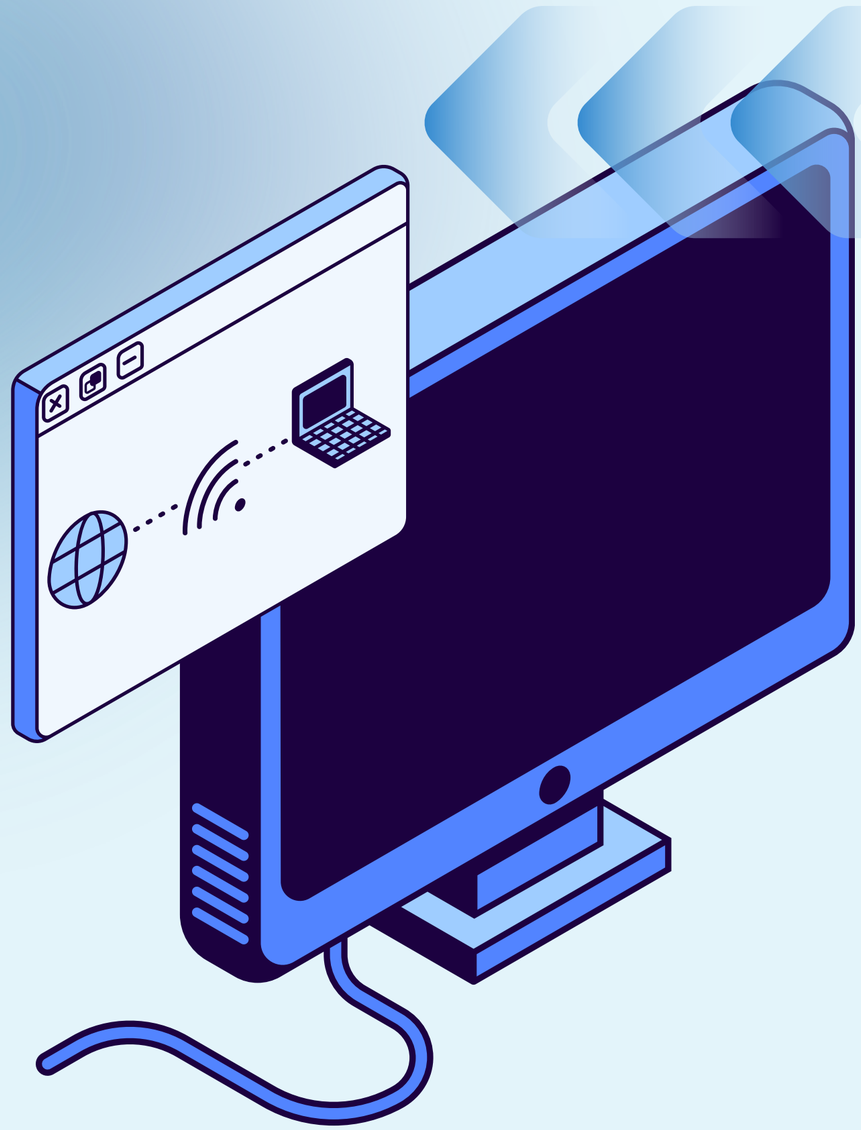
HTTP

É o protocolo fundamental da internet que permite a comunicação entre o seu navegador (como o Chrome ou Safari) e os servidores que hospedam os sites. Ele funciona como uma "ponte" que envia seus pedidos (clicar em um link) e recebe as respostas (carregar a página)



Módulo 3 — Git e fluxo de trabalho profissional

- Repositório.
- Branch.
- Pull request.
- Commit.
- Merge.
- Tags.
- .gitignore.
- README.
- Organização mínima de projeto para deploy.



Módulo 3 — Git e fluxo de trabalho profissional

O que é git?

O Git é o **sistema de controle de versão** distribuído mais utilizado no mundo, essencial para quem trabalha com desenvolvimento de software ou edição de arquivos de forma organizada. **Ele permite rastrear alterações, voltar a versões anteriores e trabalhar em equipe sem o risco de sobrescrever o trabalho de outras pessoas.**

git básico

O que é git?

O Git é uma "máquina do tempo" para o seu trabalho. Ele é um programa que salva um histórico completo de todas as alterações feitas em seus arquivos. Se você cometer um erro, pode "voltar no tempo" para qualquer versão anterior.

Problemas que o git resolve

- Versionamento de código
- Resolução de conflitos (trabalho em equipe)
- Segurança (arquivos acessíveis de qualquer local)
- histórico de modificações (quem fez o que e quando)

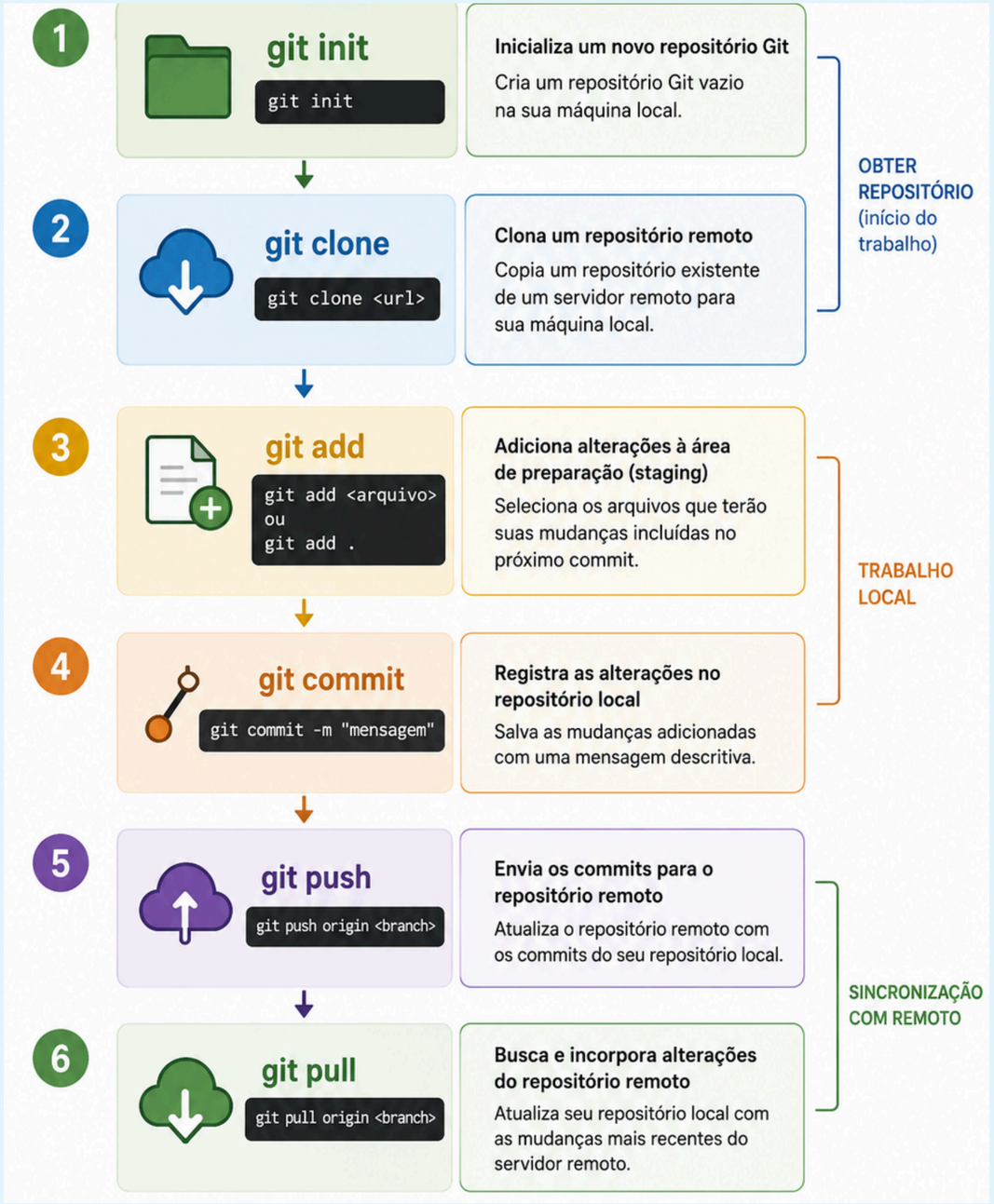
git básico

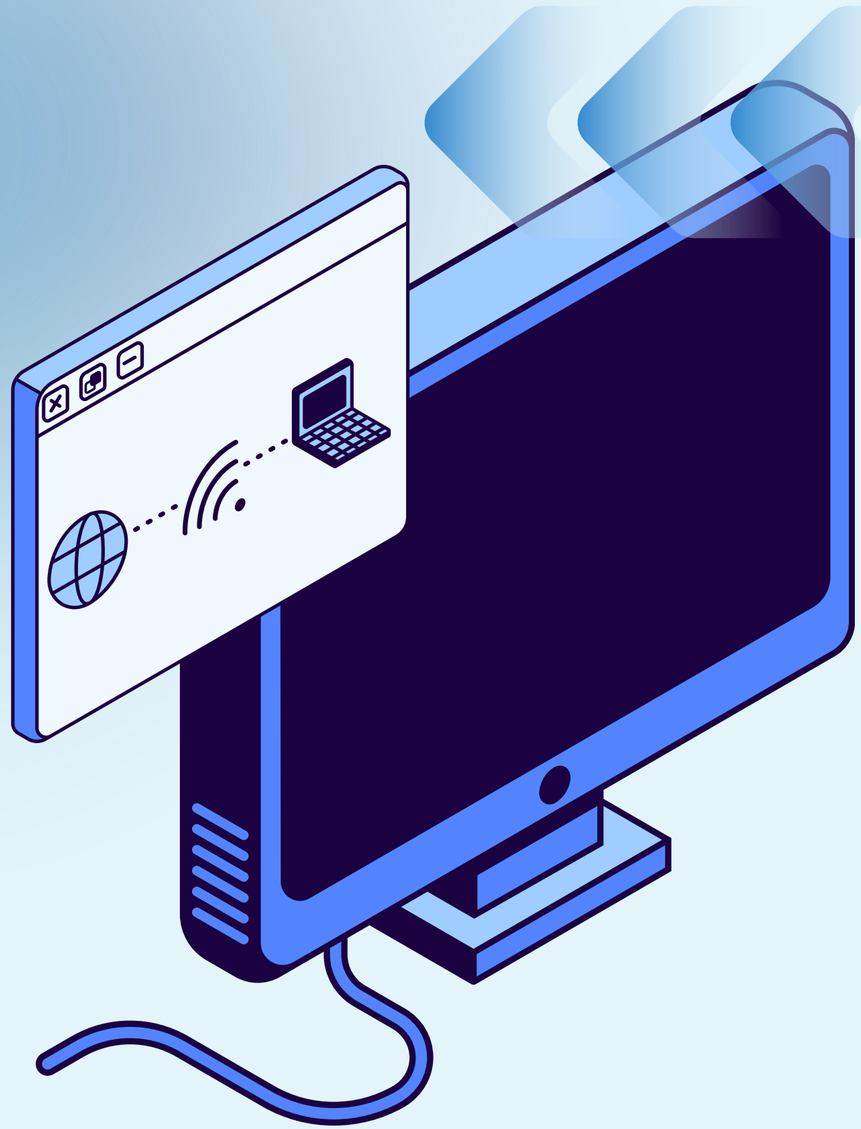
Principais comandos

COMANDOS	AÇÃO	DESCRIÇÃO
git init	O Começo	Transforma uma pasta normal do seu computador em uma "pasta vigiada" pelo Git.
git clone	A Cópia	Baixa uma cópia exata de um projeto que já existe (geralmente da internet) para o seu computador.
git status	O Relatório	Mostra um resumo do que está acontecendo na sua pasta (quais arquivos você mudou, se criou algo novo, etc.)
git add	A Seleção	Prepara os arquivos modificados para serem salvos.
git commit	O Salvamento	Salva definitivamente as alterações que você separou com o comando anterior, criando uma nova versão do seu projeto
git push	O Envio	Envia todas as suas versões salvas (commits) do seu computador para um servidor na nuvem (como o GitHub).
git pull	A Atualização	Baixa para o seu computador as últimas alterações feitas por outras pessoas que estão na nuvem.

git básico

Principais comandos

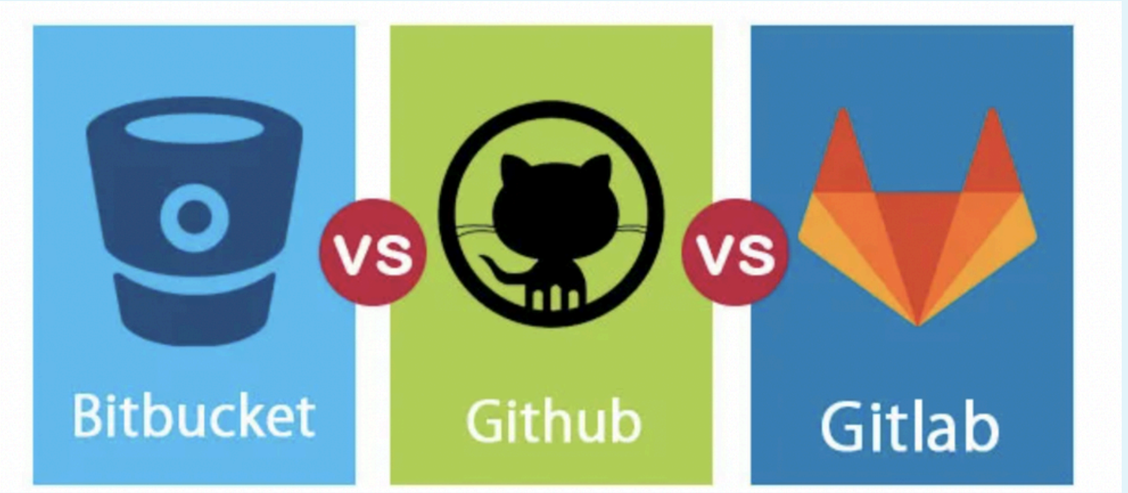




git básico

O que é o github?

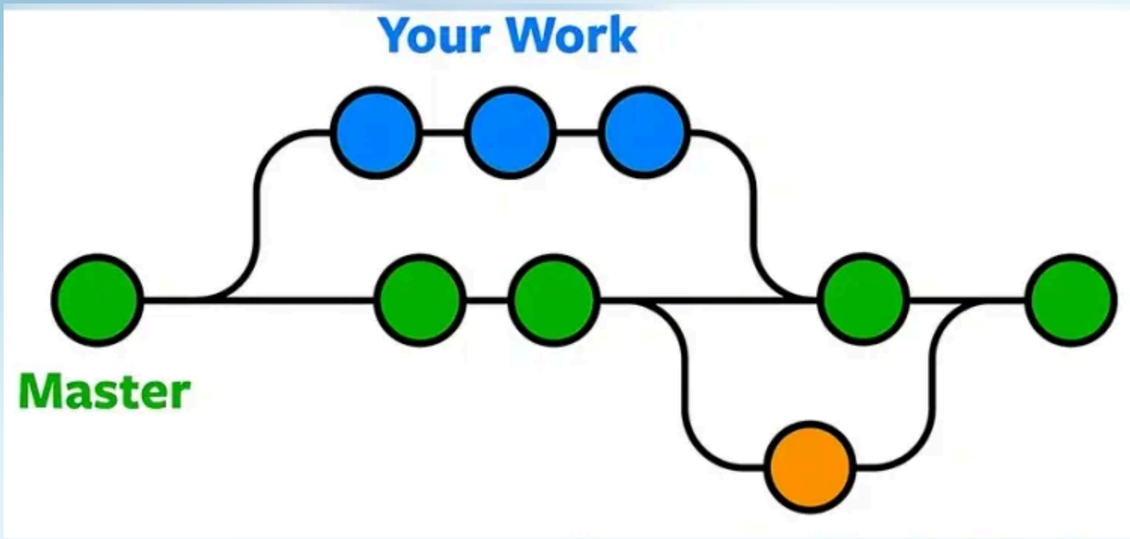
É a plataforma online (hospedada na web) onde você envia esses códigos para salvá-los e permitir que outras pessoas os vejam ou ajudem a construir.



git básico

- Criar seu repositório e fazer um commit e push.
- Clonar o repositório da imersão

git básico



$ git branch <nome-da-branch>

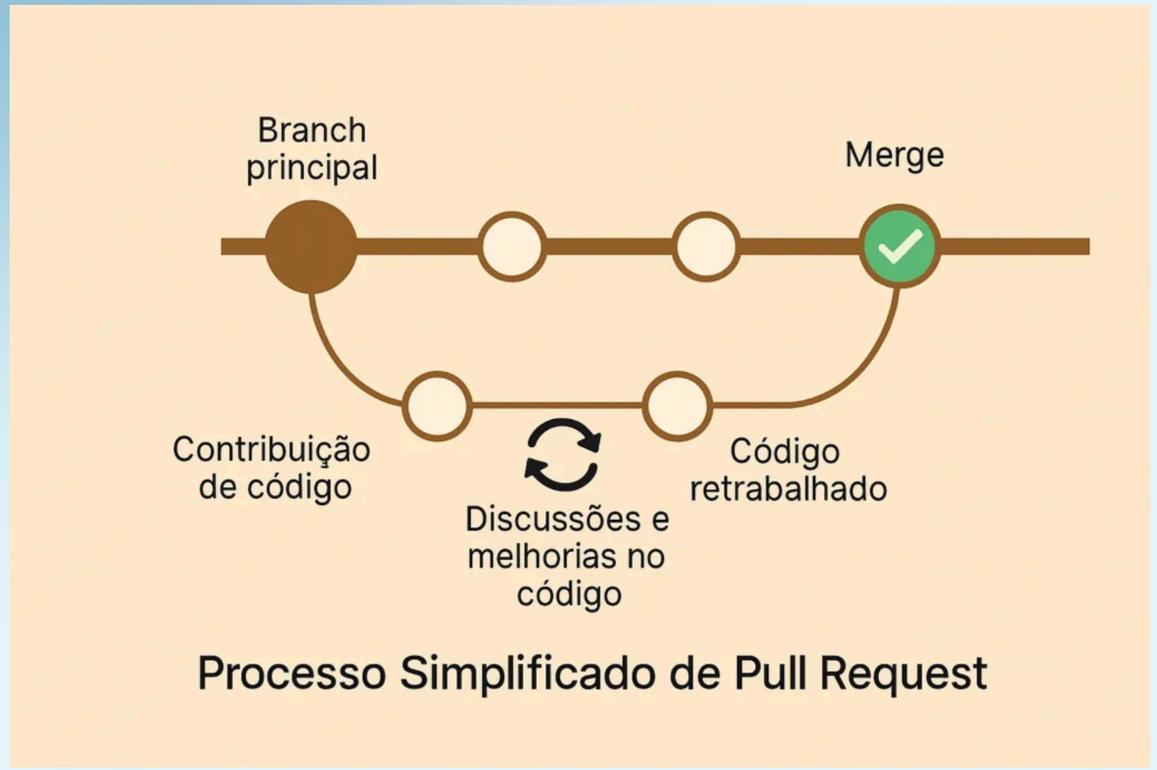
Branch

Um branch (ou ramo) no Git é uma linha independente de desenvolvimento. Na prática, ele funciona como uma ramificação do seu código, permitindo que você crie novos recursos, corrija erros ou faça testes com segurança, sem alterar ou corromper a versão principal (geralmente chamada de main ou master)

Para que serve?

- **Isolamento** - é uma linha independente de desenvolvimento.
- **Paralelismo**: Múltiplas pessoas podem trabalhar em diferentes partes do sistema ao mesmo tempo.
- **Organização**: Facilita a revisão de código antes de integrar as alterações (através de Pull Requests ou Merge Requests).

git básico



Processo Simplificado de Pull Request

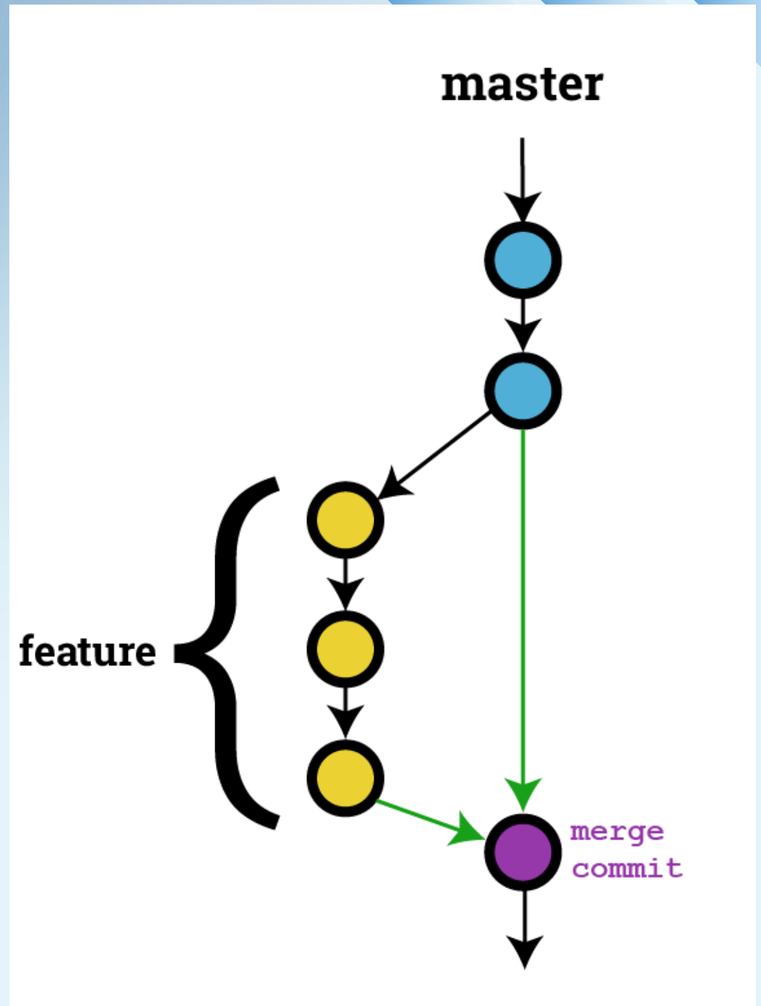
Pull Request

Pull request (PR) é uma solicitação feita para integrar as alterações de um branch em outro. No entanto, essa integração depende da análise e aprovação dos outros colaboradores do projeto. Se estiver tudo certo, as modificações são mescladas ao branch principal.
Na prática, enviar um pull request é como sugerir uma alteração em um documento compartilhado no Google Docs.

Para que serve?

- **Qualidade do Código:** Permite que outros membros da equipe revisem o código em busca de erros ou bugs antes que ele vá para o ar.
- **Compartilhamento de conhecimento**: facilita a comunicação e mantém o time todo informado sobre as novas funcionalidades que estão sendo adicionadas ao projeto.
- **Segurança**: evita edições acidentais ou indesejadas no projeto principal, uma vez que o código só é integrado após a validação da equipe

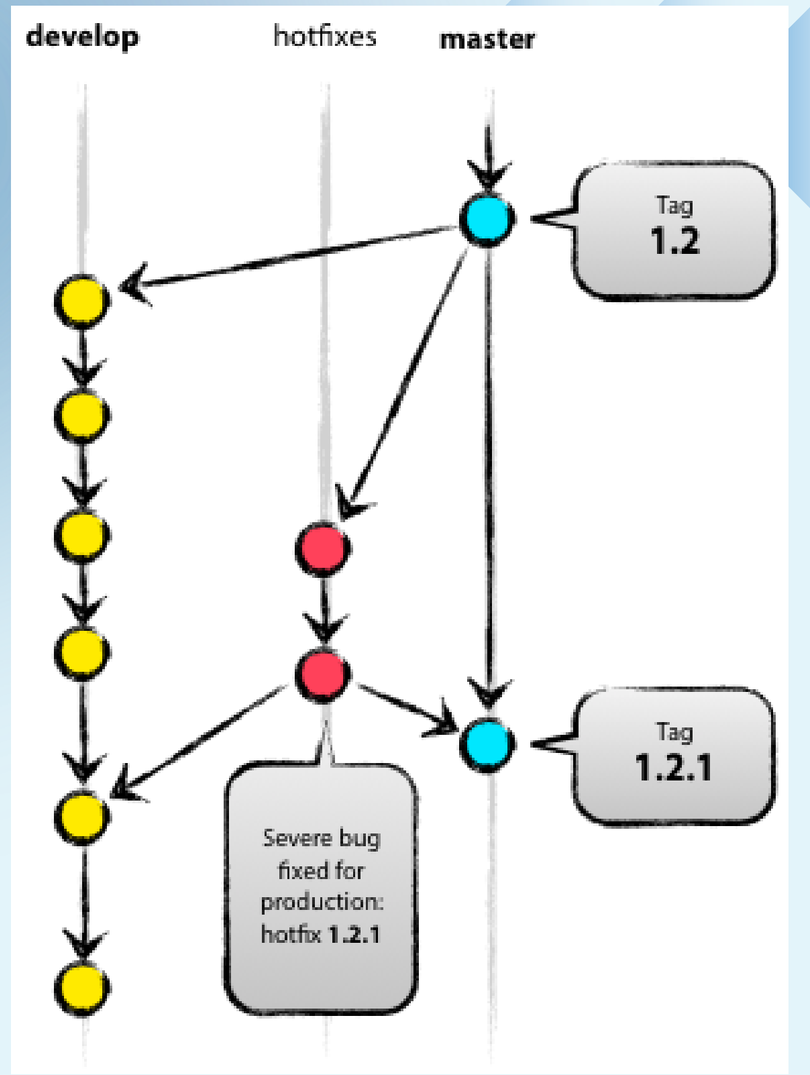
git básico



Merge

O **git merge** é o comando responsável por unificar o histórico de duas ramificações (branches) diferentes. Ele é utilizado para integrar as alterações feitas em uma branch de trabalho (como uma nova funcionalidade ou correção)

```
$ git checkout <branch-de-destino>
$ git merge <branch-com-modificacoes>
$ git log
```



git básico

tags

Tags no Git são marcadores permanentes usados para identificar pontos específicos no histórico de commits do seu repositório. Elas são amplamente utilizadas para marcar versões de lançamento de software (como v1.0, v2.1, etc.) ou marcos importantes.

$ git tag -a v1.0 -m "Versão de lançamento 1.0"
$ git push origin v1.0

git básico

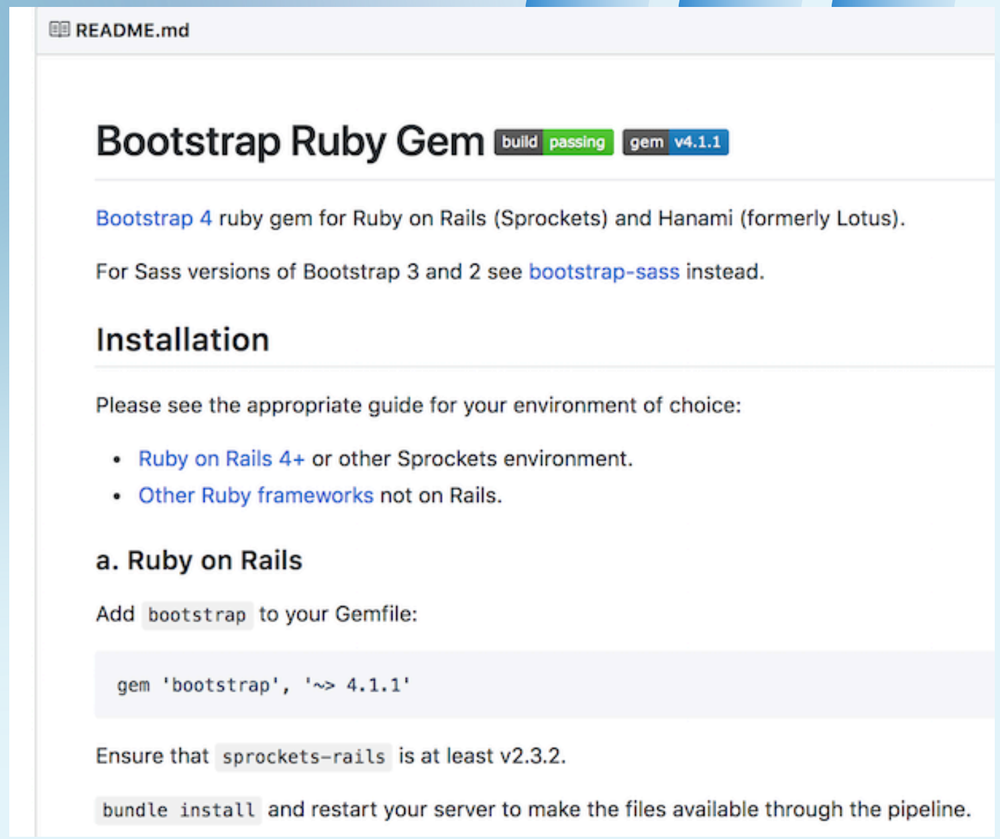
.gitignore

O arquivo .gitignore é um arquivo de texto simples que diz ao Git quais arquivos ou pastas ele não deve rastrear ou enviar para o repositório (como o GitHub ou GitLab).

Ele mantém seu projeto limpo e organizado, impedindo o versionamento de itens desnecessários ou confidenciais. Exemplos comuns do que ignorar incluem:

- **Arquivos sensíveis**: Senhas ou chaves de API (ex: arquivos .env).
- **Dependências de pacotes**: Pastas pesadas que podem ser reinstaladas depois (ex: node_modules no Node.js ou vendor no PHP).
- **Arquivos compilados e de build**: Códigos gerados automaticamente após rodar o projeto (ex: pastas dist/, build/, bin/).
- **Arquivos de configuração do sistema ou IDE**: Arquivos criados pelo seu editor de código que não interessam aos outros desenvolvedores (ex: .vscode/, .DS_Store do Mac).

git básico


README.md

Bootstrap Ruby Gem build passing gem v4.1.1

Bootstrap 4 ruby gem for Ruby on Rails (Sprockets) and Hanami (formerly Lotus).

For Sass versions of Bootstrap 3 and 2 see bootstrap-sass instead.

Installation

Please see the appropriate guide for your environment of choice:

- Ruby on Rails 4+ or other Sprockets environment.
- Other Ruby frameworks not on Rails.

a. Ruby on Rails

Add `bootstrap` to your Gemfile:

```
gem 'bootstrap', '~> 4.1.1'
```

Ensure that `sprockets-rails` is at least v2.3.2.

`bundle install` and restart your server to make the files available through the pipeline.

README.md

Um **README** (geralmente nomeado como **README.md**) é o arquivo de documentação principal de um projeto armazenado no Git ou hospedado em plataformas como o GitHub. Ele funciona como o manual de instruções ou a página de apresentação do seu código, ajudando qualquer pessoa a entender rapidamente o que é o projeto e como utilizá-lo

Módulo 4 — Docker

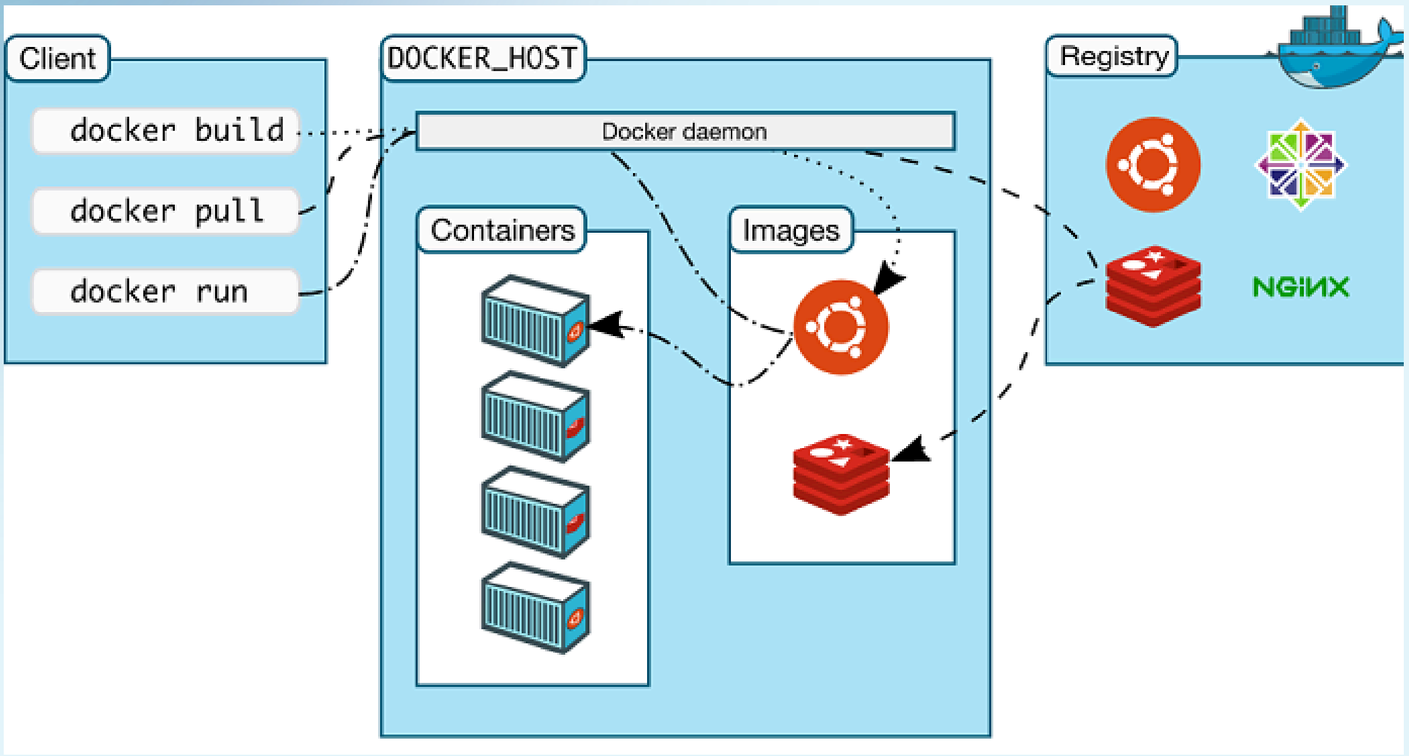
- Image vs container
- Dockerfile
- Build
- Run
- Exposição de portas
- Volumes
- Variáveis de ambiente
- Logs do container
- Erros comuns
- Boas práticas

Módulo 4 — Docker

O **Docker** é uma plataforma de código aberto que permite criar, implantar e gerenciar aplicativos isolados em pacotes chamados **contêineres**. Ele resolve o famoso problema de incompatibilidade, garantindo que um programa rode exatamente da mesma forma no seu computador de desenvolvimento e no servidor de produção.

O Docker é uma ferramenta que funciona como um "contêiner de navio" digital. Ele empacota um programa e todas as coisas que ele precisa para funcionar, garantindo que esse programa rode exatamente da mesma forma em qualquer computador, seja no seu notebook ou no servidor da empresa.

Módulo 4 — Docker
Container, Image e Registry



- **Imagem** é o pacote pronto da aplicação, com código, dependências e configurações.
- **Container** é a aplicação rodando a partir dessa imagem.
- **Registry** é (ou registro) é um repositório centralizado onde as imagens são armazenadas, organizadas e de onde podem ser baixadas.
  Exemplo: dockerhub, ECR, gitlab registry

Módulo 4 — Docker
Container, Image e Registry

- **Rodando um container docker a partir de uma image**
  $ docker run -it ubuntu bash

- **Para sair de um container em execução sem encerrar o processo:**
  $ CTRL + P e CTRL + Q

- **Ver containers rodando:**
  $ docker ps

- **Voltar ao container original:**
  docker attach *<id_ou_nome_do_container>*

- **Abrir um novo terminal no container:**
  $ docker exec -it *<id_ou_nome_do_container>* bash

- **Remover um container**
  $ docker rm -f *<id_ou_nome_do_container>*

Módulo 4 — Docker
Container, Image e Registry

- **Parar um container que está rodando.**
  $ docker stop <id_ou_nome_do_container>

- **Listar todos os containers:**
  $ docker ps -a

- **Listar as imagens docker disponíveis na sua máquina:**
  $ docker images

Módulo 4 — Docker
Container, Image e Registry

- **Parar um container que está rodando.**
  $ docker stop <id_ou_nome_do_container>

- **Listar todos os containers:**
  $ docker ps -a

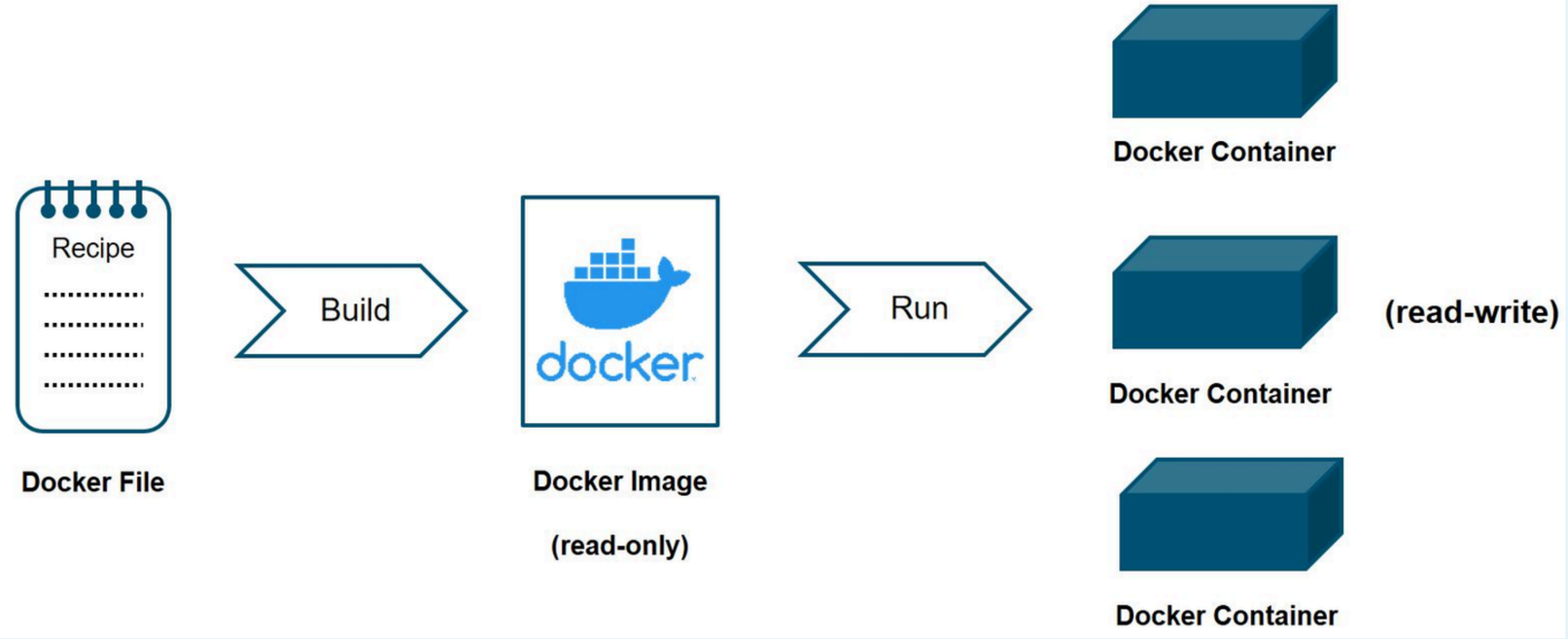
- **Listar as imagens docker disponíveis na sua máquina:**
  $ docker images

Módulo 4 — Docker

Criando uma imagem docker personalizada

O que é um Dockerfile?

Um Dockerfile é um script que contém instruções para construir uma imagem Docker. Ele define a imagem base, configura variáveis de ambiente, instala software e configura o contêiner para uma aplicação ou serviço específico.

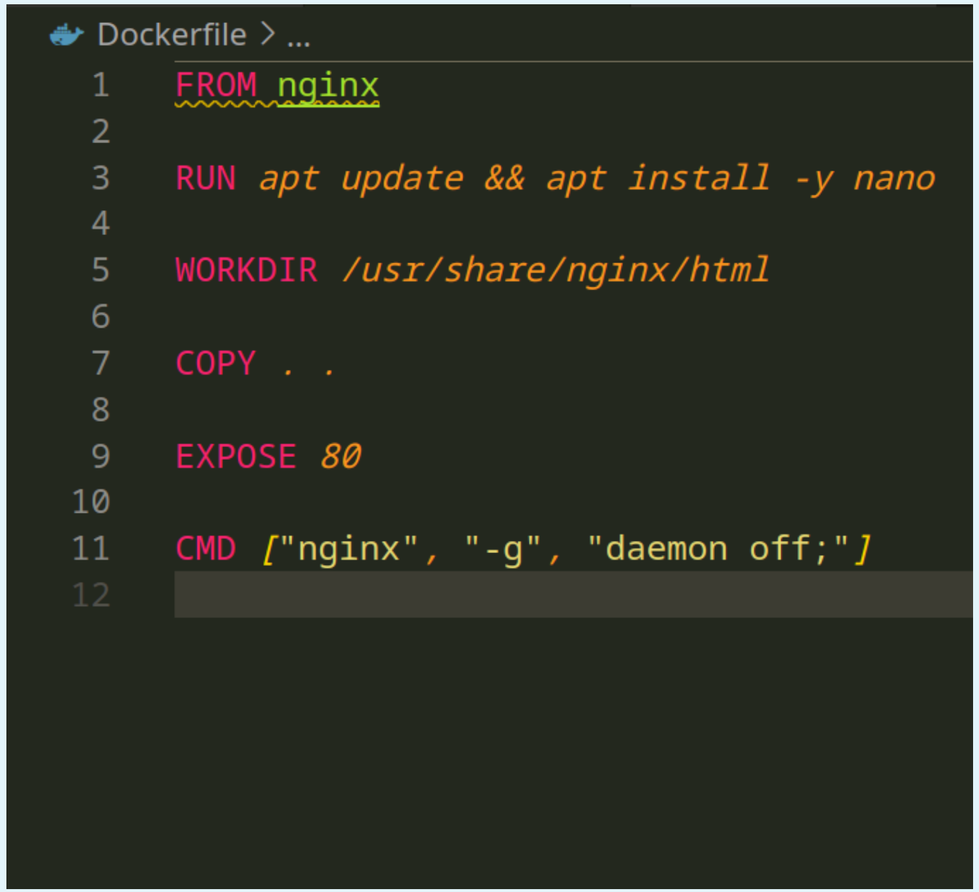


Módulo 4 — Docker

Criando uma image docker personalizada

🛠️ Passo a Passo para Criar sua Imagem

1. **Crie o Dockerfile:** Na pasta do seu projeto, crie um arquivo de texto sem extensão chamado Dockerfile.
2. **Defina a Imagem Base (FROM):** Toda imagem precisa de uma base, como o ubuntu ou uma imagem oficial de linguagem (python, node).
3. **Configure o Diretório (WORKDIR):** Defina a pasta onde sua aplicação ficará dentro do contêiner.
4. **Copie os Arquivos (COPY):** Transfira o código-fonte da sua máquina para dentro do contêiner.
5. **Instale Dependências (RUN):** Execute comandos para instalar os pacotes necessários.
6. **Defina a Porta (EXPOSE):** Informe qual porta o contêiner vai escutar.
7. **Comando de Inicialização (CMD):** Determine o comando que iniciará a aplicação ao rodar o contêiner.
8. **Comando para build da image docker:**
   $ docker build -t *<nome-da-image>* **.**



Módulo 4 — Docker

Criando uma image docker personalizada

🛠️Enviando a image para o registry (dockerhub)

1 - Comando para build da image docker:

$ docker build -t <sua-conta-dockerhub>/*<nome-da-image>* **.**

2 - Login no registry

$ docker login -u *<username>*

3 - Enviar image para o registry

$ docker push

4 - Agora é congerir no registry se a imagem foi salva

- IMPORTANTE - Não utilizar a tag latest, isso evita problemas futuros com quebras inesperadas de versões

```
~/poc/nginx-test
 docker login
Authenticating with existing credentials... [Username: jhousyfran]

 Info → To login with a different account, run 'docker logout' followed by 'docker login'


Login Succeeded

~/poc/nginx-test
 docker build -t jhousyfran/app-imersao-html .
[+] Building 0.1s (9/9) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 177B
 => [internal] load metadata for docker.io/library/nginx:latest
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [1/4] FROM docker.io/library/nginx:latest
 => [internal] load build context
 => => transferring context: 63B
 => CACHED [2/4] RUN apt update && apt install -y nano
 => CACHED [3/4] WORKDIR /usr/share/nginx/html
 => CACHED [4/4] COPY . .
 => exporting to image
 => => exporting layers
 => => writing image sha256:bca692757af98b60e8778f9e457dbf84cddc3bc95ff9a7a909afd4d2e1bb9222
 => => naming to docker.io/jhousyfran/app-imersao-html

~/poc/nginx-test
 docker push jhousyfran/app-imersao-html:latest
The push refers to repository [docker.io/jhousyfran/app-imersao-html]
a974d8c71ec8: Pushed
5f70bf18a086: Mounted from jhousyfran/jvb
0d6841486664: Pushed
121b6a4dad00: Mounted from library/nginx
db054fb45c4c: Mounted from library/nginx
57b80f15e540: Mounted from library/nginx
e8832fb2fc12: Mounted from library/nginx
493aa43406a6: Mounted from library/nginx
af29206eb654: Mounted from library/nginx
f4f8b983b714: Mounted from library/nginx
latest: digest: sha256:e12dd2300694fddcb6afbc0438246e16a4ab7a37f2e02aeb33cfb5fc3eebeb75 size: 2405
```