

# Ambiente de Dados e Operações - DataOps

Versionamento, Git e CI/CD

---

Professor MSc. Fernando Sousa

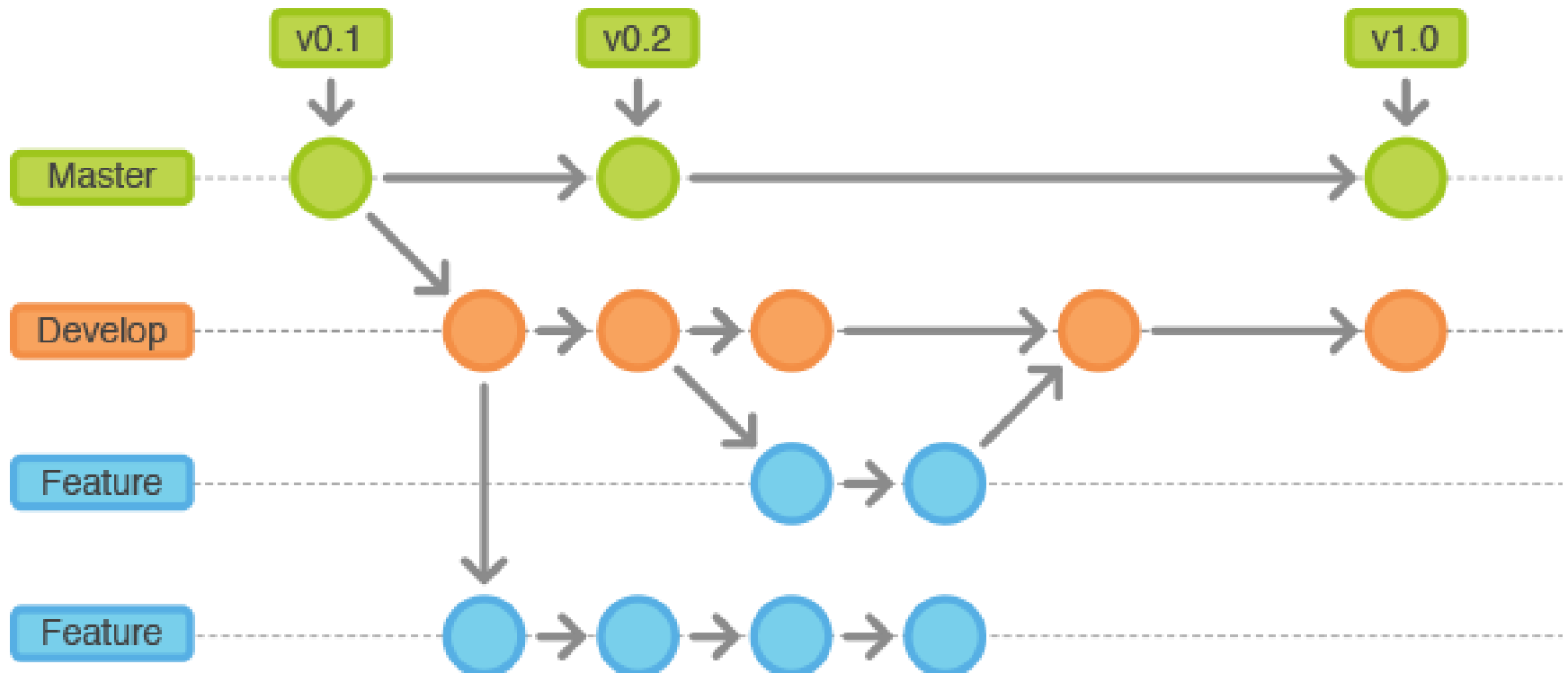
# Controle de Versão

---

- Um sistema de controle de versão é um sistema que grava todas as alterações feitas em um ou mais arquivos ao longo do tempo.
- Dessa forma, é possível recuperar uma versão anterior quando for necessário
- Cada versão, às vezes chamada de revisão, pode ser comparada com outras, restauradas e às vezes integradas (*merge*)
- Usualmente existe uma versão principal (master, main ou *trunk*) na qual as alterações são integradas para a implantação

# Controle de Versão

- Exemplo de como controle de versão funciona



Fonte: <https://i.pinimg.com/originals/2a/5d/5c/2a5d5c1d5f42a3d0fd712ae7e4b23824.png>

# Controle de Versão

---

- Quando uma nova funcionalidade for desenvolvida uma *branch* (ramo) é criada a partir da versão principal
  - Mais de uma funcionalidade pode ser desenvolvida ao mesmo tempo sem interferência
  - Eventualmente podem ser integradas na versão principal, em tempos diferentes

# Controle de Versão

---

- Bastante útil em data analytics
  - Facilita o trabalho em equipe
  - Facilita a recuperação de arquivos
  - Facilita a reversão da versão
  - Facilita a implantação em diferentes ambientes
- Usar um sistema de controle de versão ajuda na implementação de DataOps
  - Versionamento dos códigos
  - Criação de branches e integração na versão principal (branch & merge)

# Git

---

- Git é um sistema de controle de versão distribuído, gratuito e de código aberto
- Existem implementações comerciais do Git, que facilitam o uso
  - Github
  - BitBucket
  - Gitlab
  - Amazon CodeCommit
- “Estado da arte” do controle de versão

# Git - Características

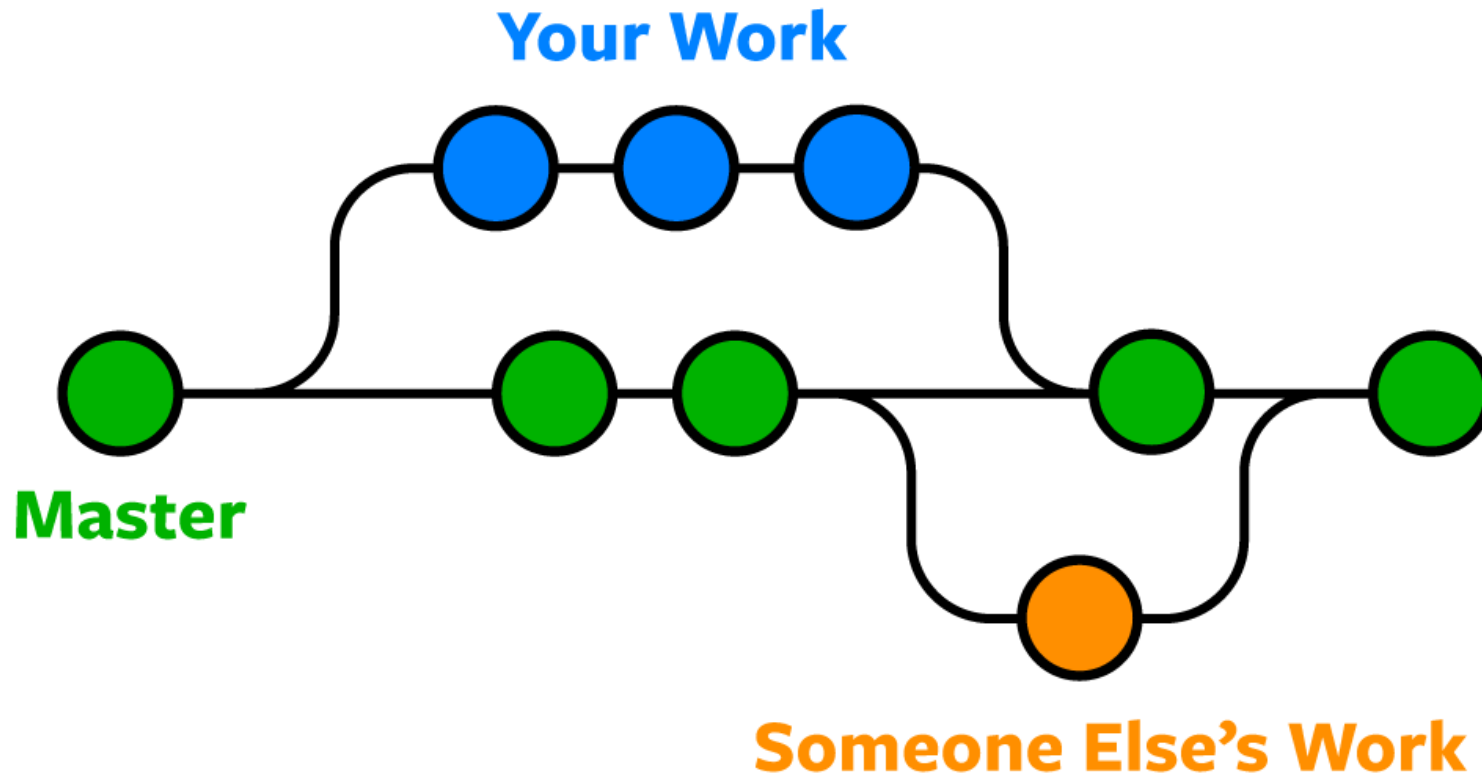
---

- Branching and merging
  - Criação de múltiplas branches locais totalmente independentes
  - Ideal para testar e inovar
    - Não deu certo, joga fora, sem interferir no código principal ou em outras branches
  - Ideal para desenvolver novas funcionalidades
  - Ideal para criar ramos para diferentes ambientes (produção, desenvolvimento, testes...)
  - Quando finalizar o trabalho, pode-se juntar na versão principal

# Git - Características

---

- Branching and merging





# Git - Características

---

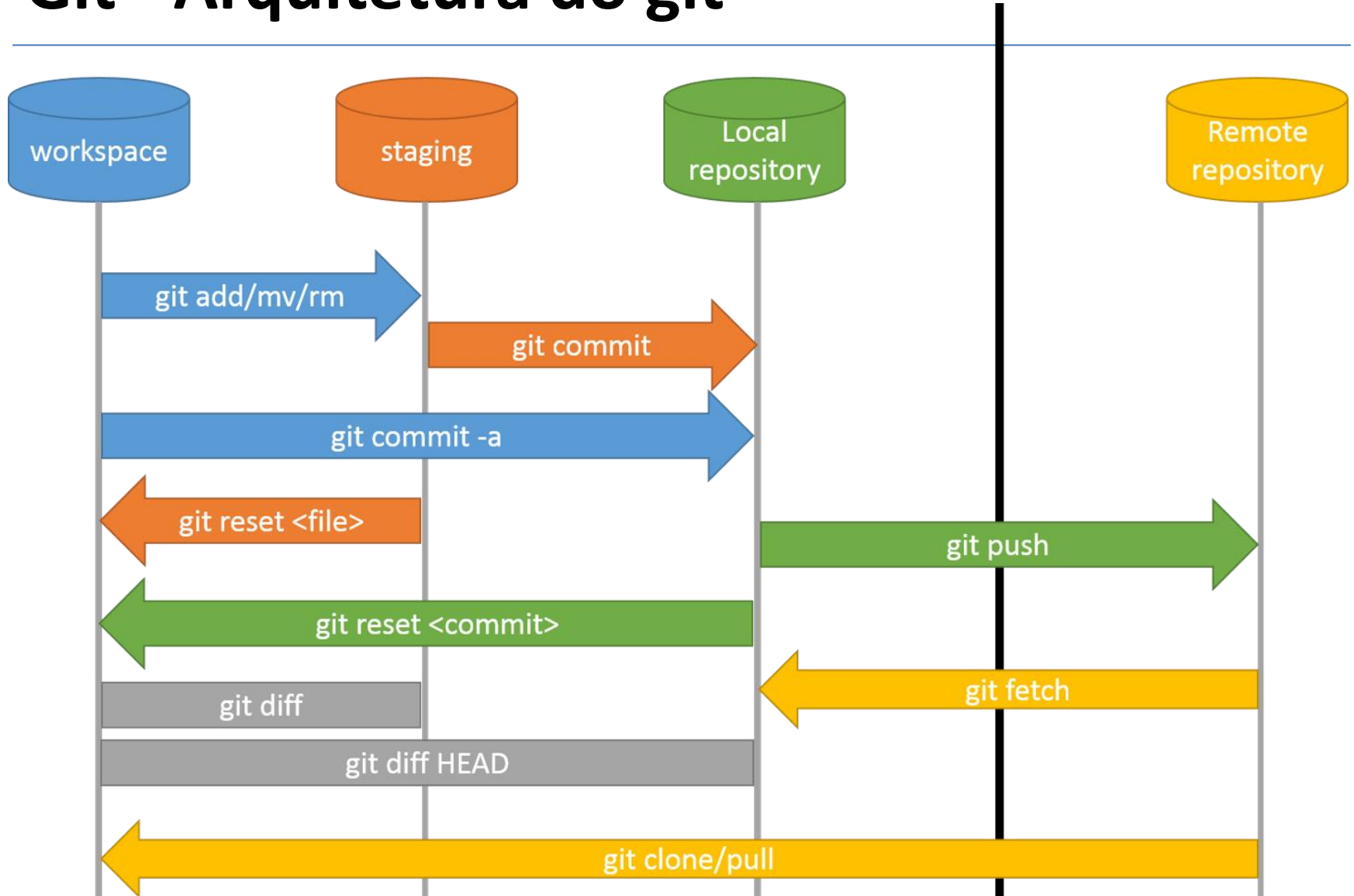
- Compacto e rápido, em comparação com outros sistemas de controle de versão
  - Não precisa de comunicação com o repositório remoto para criar as branches
    - Branches locais
  - Só comunica com o repositório remoto quando tiver finalizado (push)
- Distribuído
  - Todo o repositório é copiado localmente (clone)
  - Todos os usuários tem um backup completo do repositório

# Git - Características

---

- Garantia de dados
  - Garantia de que os dados baixado são exatamente os que foram enviados
  - Um commit não pode ser alterado
    - Um novo commit gera um novo commit ID
    - Ao recuperar os dados do commit ID, há a garantia que ele não foi alterado depois
- Área de Staging
  - Área intermediária antes de enviar as alterações para o repositório local (commit)
  - Possibilita adicionar somente alguns arquivos para fazer o commit

# Git - Arquitetura do git



# Git – Branch

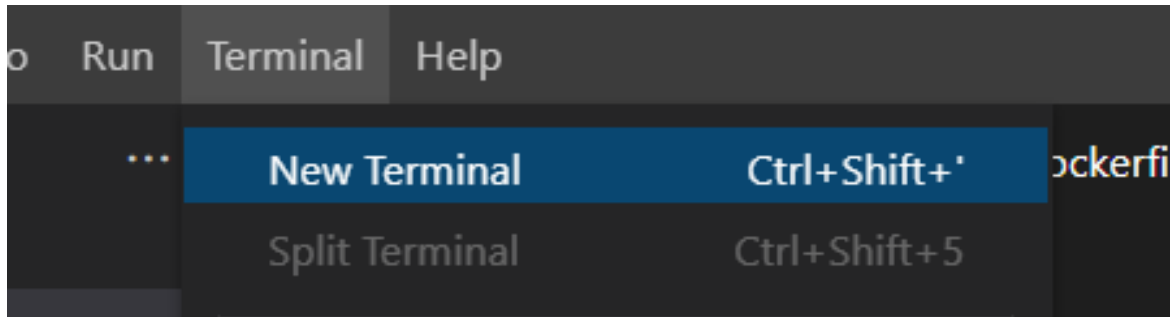
---

- Ramo criado para fazer alterações nos arquivos
- Ramos são totalmente independentes
- As alterações podem ser integradas em outros ramos quando solicitado
- Podem ser criadas quantas branches forem necessárias
- A remoção de uma branch não interfere em outras

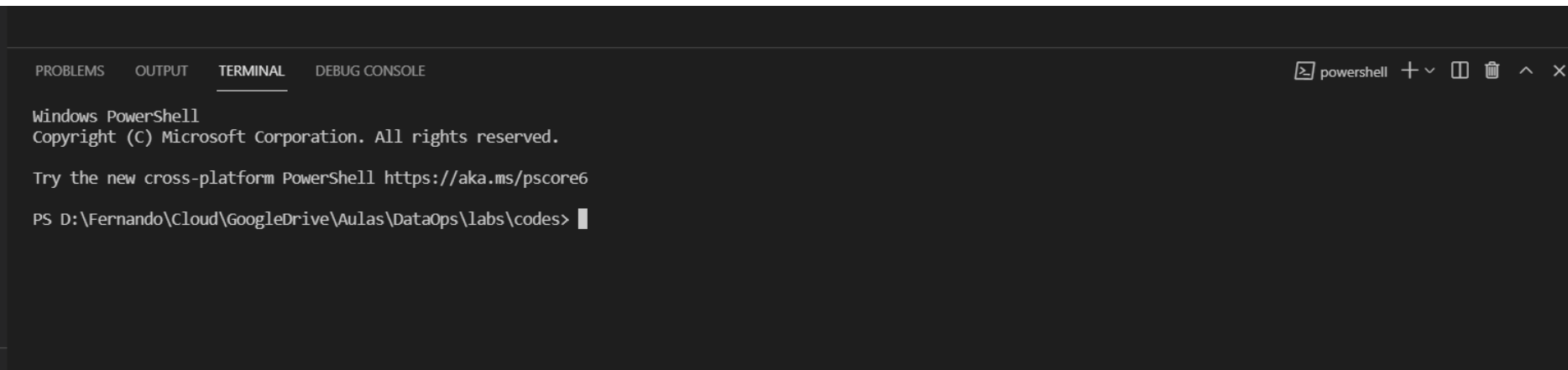
# Git - Instalação

---

- Abra o VSCode e abra um novo terminal
  - Menu Terminal → New Terminal



- O terminal será aberto na parte inferior do VSCode



# Git - Instalação

---

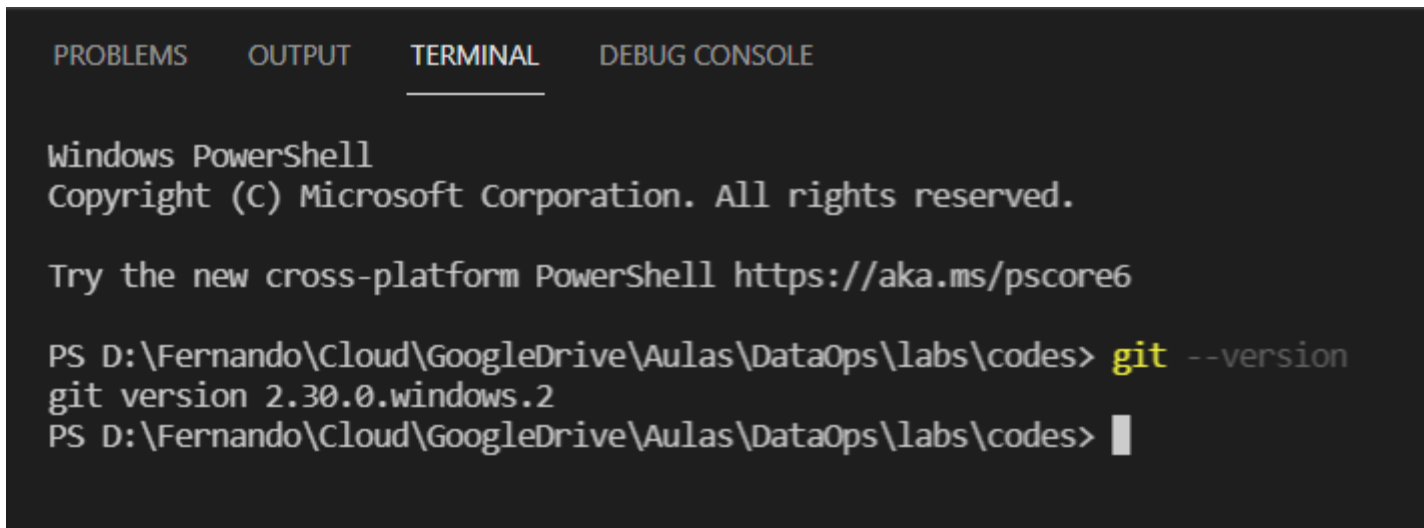
- Importante!
- O terminal sempre vai abrir na pasta que está aberta no VSCode
- Verifique sempre se está executando os comando a partir da pasta correta do terminal

# Git - Instalação

---

- No terminal execute o seguinte comando para verificar se o Git está instalado

**git --version**



```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS D:\Fernando\Cloud\GoogleDrive\Aulas\DataOps\labs\codes> git --version
git version 2.30.0.windows.2
PS D:\Fernando\Cloud\GoogleDrive\Aulas\DataOps\labs\codes> █
```

# Git - Instalação

---

- Se o terminal mostrar a versão do Git, ele já está instalado
- Caso o resultado seja parecido com o seguinte:
  - 'git' não é reconhecido como um comando interno ou externo, um programa operável ou um arquivo em lotes
- Entre em <https://git-scm.com/download/win> e baixe a versão compatível com o seu sistema operacional.
- Siga as instruções de instalação
- Quando terminar, abra um novo terminal no VSCode e teste o comando novamente
  - Se ainda não funcionar, reinicie o VSCode



# GitHub

---

- GitHub é uma implementação do Git na nuvem
- Possibilita a criação de repositórios públicos e privados
  - Empresas gostam de ver o GitHub dos candidatos
- Acesse <https://github.com> e faça seu cadastro clicando em “Sign up”, caso ainda não tenha, e siga os passos
- Quando terminar, acesse novamente e veja se já está autenticado
  - Se não clique em “Sign in”
- Caso já tenha cadastro, basta fazer o login

# Github - Repositório

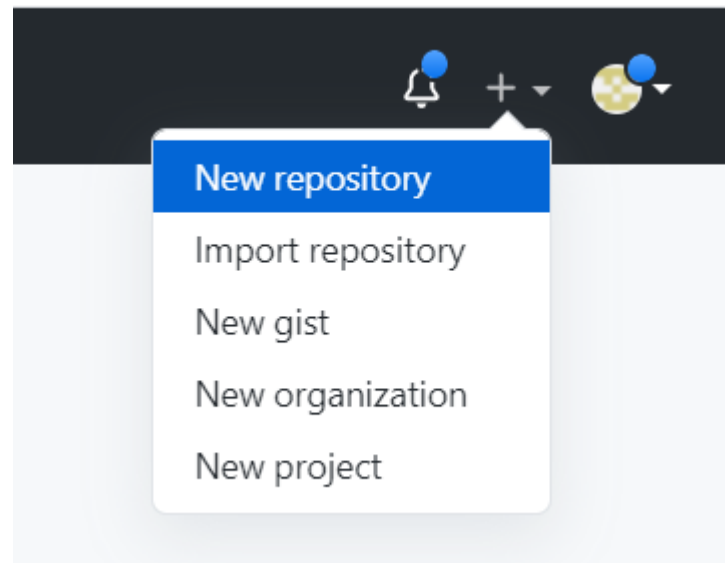
---

- Unidade principal de um sistema de controle de versão
- Contém os arquivos de uma aplicação que serão versionados

# Github - Repositório

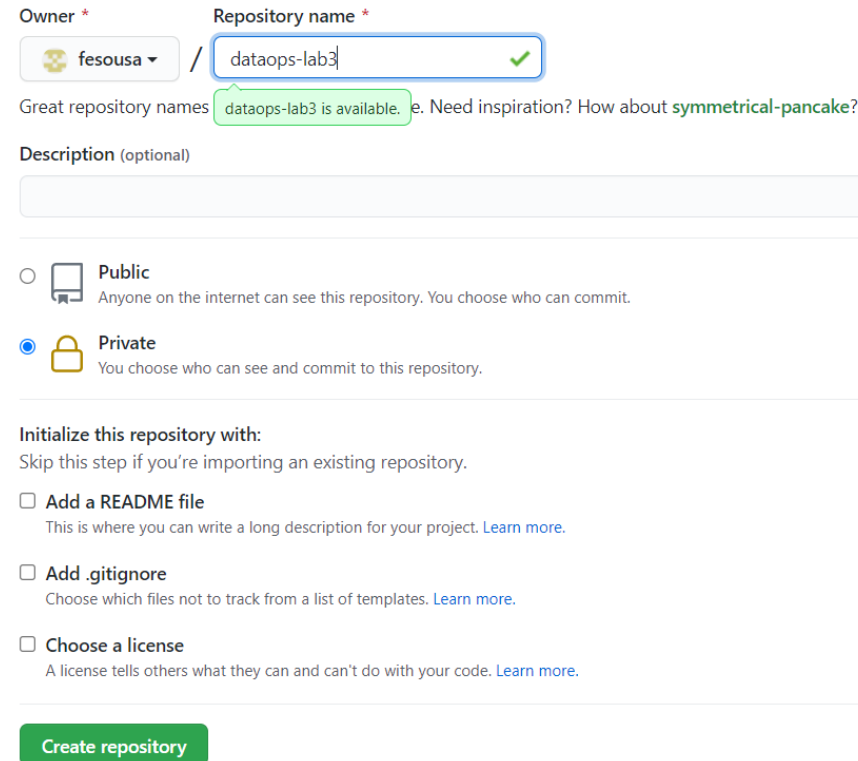
---

- Vamos criar um repositório para o próximo laboratório (lab3)
- Depois de autenticar no Github, clique no “+” na barra superior a direita e depois em “New repository”



# Github - Repositório

- Coloque o nome do repositório (Repository name)
  - dataops-lab3
- Escolha a opção “Private” (repositório privado)
- Clique em “Create repository”
- Um repositório vazio será criado
- O Github mostrará um tela com alguns comandos para adicionar arquivos no repositório



The screenshot shows the GitHub 'Create repository' interface. At the top, the 'Owner' is set to 'fesousa' and the 'Repository name' is 'dataops-lab3', which is highlighted with a green checkmark. Below this, a message states 'Great repository names' and 'dataops-lab3 is available.' followed by a suggestion: 'e. Need inspiration? How about [symmetrical-pancake?](#)'. A 'Description (optional)' text area is present below. The visibility options are 'Public' (unchecked) and 'Private' (checked with a blue radio button). The 'Private' option is accompanied by a lock icon and the text 'You choose who can see and commit to this repository.' Under the heading 'Initialize this repository with:', there is a note to 'Skip this step if you're importing an existing repository.' Three checkboxes are listed: 'Add a README file' (unchecked), 'Add .gitignore' (unchecked), and 'Choose a license' (unchecked). Each checkbox has a brief description and a 'Learn more' link. At the bottom, a green 'Create repository' button is visible.

Owner \* / Repository name \*

Great repository names [dataops-lab3 is available.](#) e. Need inspiration? How about [symmetrical-pancake?](#)

Description (optional)

☐ Public  
Anyone on the internet can see this repository. You choose who can commit.

☒ Private  
You choose who can see and commit to this repository.

Initialize this repository with:  
Skip this step if you're importing an existing repository.

☐ Add a README file  
This is where you can write a long description for your project. [Learn more.](#)


☐ Add .gitignore  
Choose which files not to track from a list of templates. [Learn more.](#)

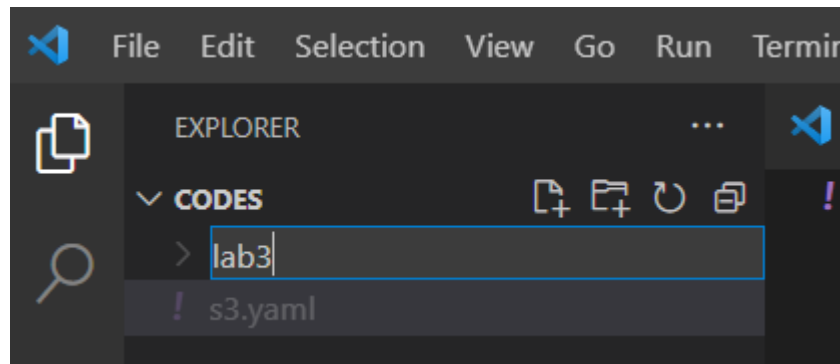
☐ Choose a license  
A license tells others what they can and can't do with your code. [Learn more.](#)

Create repository

# Github – Criar repositório local

---

- O repositório criado está vazio
- É preciso criar um repositório local (no seu computador) e enviar para o repositório do Github
- Nos exemplos de aula vamos estruturar um repositório em uma pasta dentro do projeto do VSCode
- Volte ao VSCode e crie uma pasta com o nome “lab3” clicando no ícone  ao lado da pasta principal do projeto

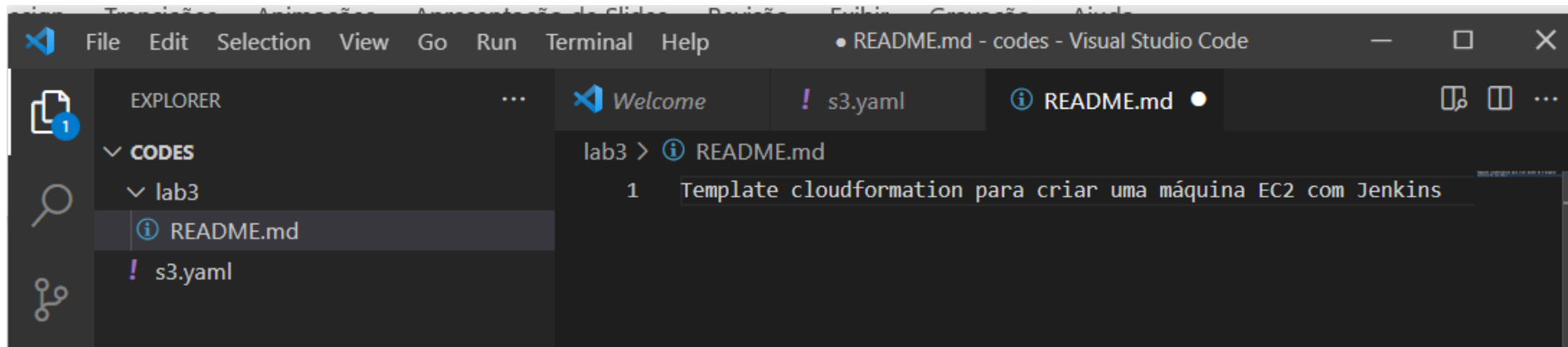


# Github – Criar repositório local

---

- Na nova pasta, crie um arquivo chamado README.md
- Coloque o seguinte conteúdo:

Template cloudformation para criar uma máquina EC2 com Jenkins



# Github – Criar repositório local

---

- Volte ao terminal do VSCode e execute os seguintes comandos
  - Abrir pasta lab3  
`cd lab3`
  - Iniciar um novo repositório na pasta  
`git init`
  - Adicionar os arquivos da pasta na área de staging  
`git add .`
  - Criar uma versão das alterações no repositório local, com uma descrição das alterações  
`git commit -m 'commit inicial'`

# Github – Criar repositório local

---

- Volte ao terminal do VSCode e execute os seguintes comandos
  - Criar o ramo principal (master)  
`git branch -M master`
  - Adicionar a URL do repositório remoto  
`git remote add origin <url repositório>`
    - Você consegue a url do seu repositório ao acessar o repositório no Github. Ela termina com .git
  - Enviar as alterações para o repositório  
`git push -u origin master`



# Github – Criar repositório local

---

- Ao executar o comando push, uma janela vai abrir para fazer a autenticação no Github
- Clique em “Sign in with your browser” para fazer a autenticação
  - Se já estiver autenticado no Github pelo navegador, a autorização é automática
  - Isso só precisa ser feito uma vez
- Volte á página do repositório no GitHub, atualize e veja que o arquivo está lá

# Git – Comando Básicos

---

- `git clone [url_repositório]`
  - Clonar um repositório remoto localmente
- `git init`
  - Criar um novo repositório local
- `git add [arquivo]`
  - Adicionar um novo arquivo para commit (staging)
  - `git add .` Adiciona todos os arquivos
- `git commit -m “mensagem de commit”`
  - Envia as alterações adicionadas para o repositório local
- `git branch [nome_branch]`
  - Cria uma nova branch a partir da atual

# Git – Comando Básicos

---

- `git checkout [nome_branch]`
  - Muda a branch que está trabalhando
- `git remote add origin [url_repositório]`
  - Especificar qual o repositório remoto vinculado ao repositório local
- `git pull`
  - Atualizar repositório local com as últimas alterações do repositório remoto
- `git merge [branch]`
  - Integra a branch na branch que está trabalhando

# Git – Comando Básicos

---

- Faça o teste
  - Crie um novo arquivo **ec2-jenkins.yaml** na pasta lab3
  - Execute os comandos no terminal

```
git add ec2-jenkins.yaml
```

```
git commit -m 'novo arquivo ec2-jenkins.yaml'
```

```
git push -u origin master
```

- Veja no Github o novo arquivo

# Trabalhando com branches

---

- Durante o desenvolvimento e evolução de uma aplicação é comum a criação de branches
  - Desenvolver funcionalidades em paralelo
  - Não atrapalhar o trabalho dos outros da equipe
  - Testar soluções
  - Não comprometer o código principal (que está funcionando)
- Quando o desenvolvimento de uma branch é finalizado, testado e aprovado, o código é integrado (merge) na branch principal para ir para produção

# Trabalhando com branches

---

- Crie uma nova branch no seu repositório do lab3
- No terminal, digite o comando  
`git branch develop`
  - Cria uma nova branch com o conteúdo da branch que está trabalhando
- Digite o comando abaixo para alterar a branch  
`git checkout develop`
- Verifique a branch atual  
`git status`
  - Você deverá ver uma mensagem parecida com essa:

```
On branch develop
nothing to commit, working tree clean
```

# Trabalhando com branches

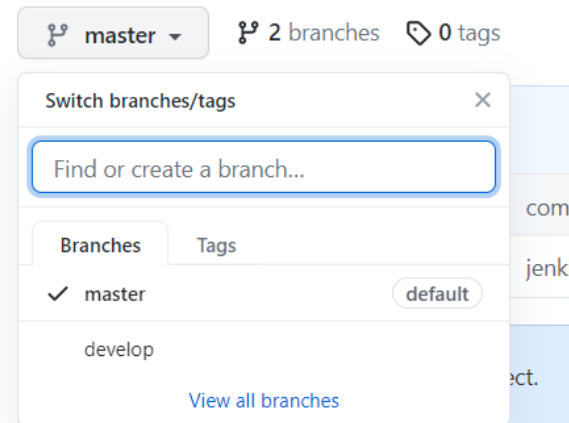
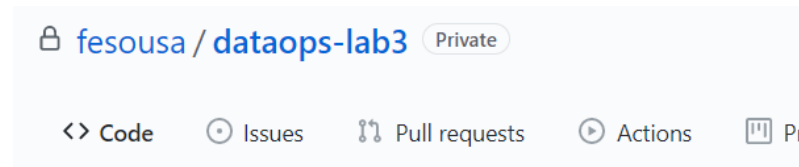
---

- Envie a nova branch para o repositório

```
git push origin develop
```

- Abra o repositório no Github e verifique que agora há duas branches

- master
- develop



# Pull request

---

- Quando uma atualização em uma branch é testada e aprovada, ela pode ser integrada na branch principal para ser colocada em produção
  - Chamado de ***pull request***
- Coloque um comentário no arquivo **ec2-jenkins.yaml** e envie para o repositório da branch develop

# Template CloudFormation para provisionar uma instância EC2 com jenkins

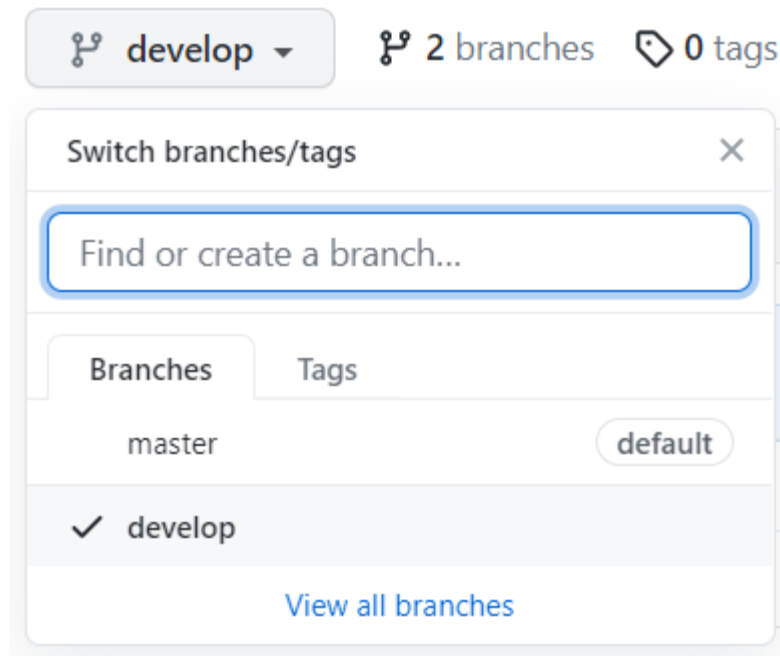
- Lembre-se dos comandos
  - git add .
  - git commit -m 'comentário'
  - git push origin develop



# Pull request

---

- Abra o repositório no Github
- Selecione a branch develop




# Pull request

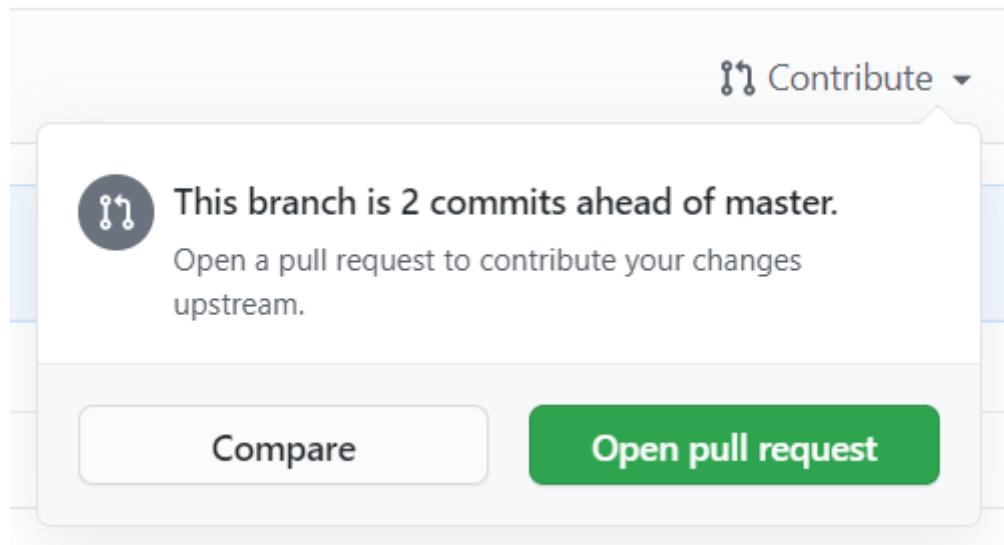
---

- Procure pela opção “contribute”

This branch is 2 commits ahead of master.

 Contribute ▼

- Clique nela e clique no botão “Open pull request”



# Pull request

---

- Verifique quais as branches selecionadas no pull request
  - Queremos fazer um pull request da develop (compare) para a master (base)



base: master ▼



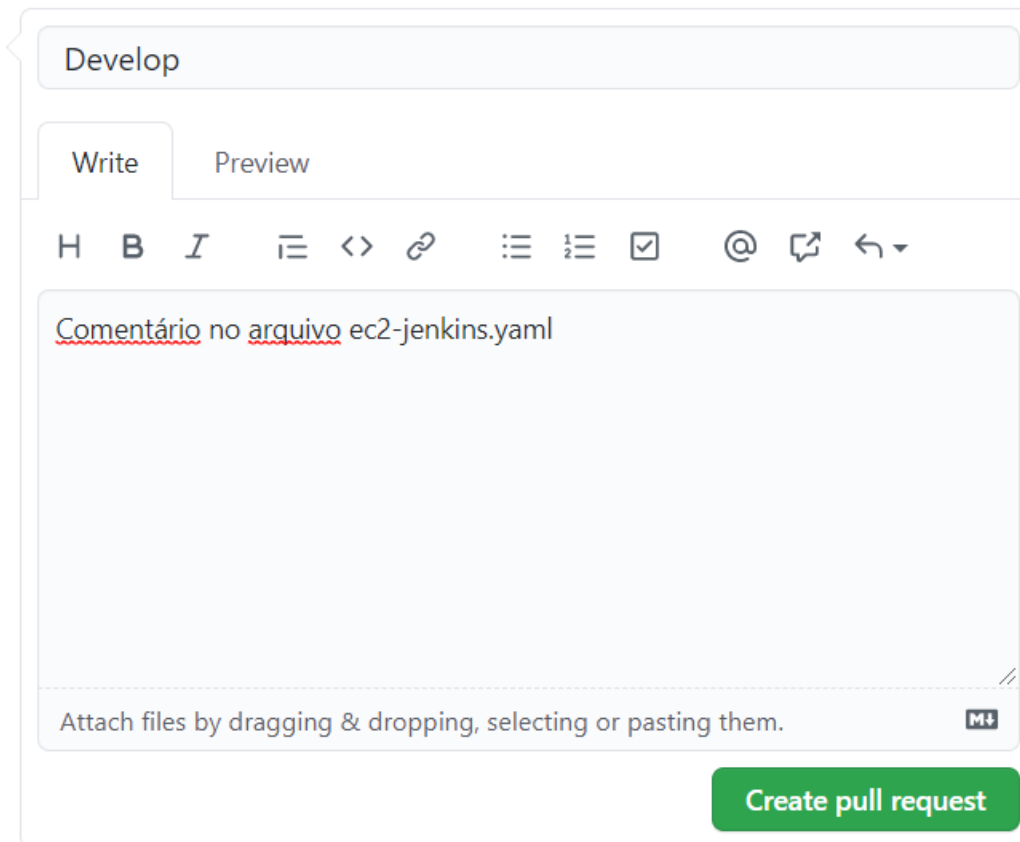
compare: develop ▼

✓ **Able to merge.** These branches can be automatically merged.

# Pull request

---








- Coloque um comentário para documentar a alteração e clique em “Create pull request”




The screenshot shows a web interface for creating a pull request. At the top, a dropdown menu is set to 'Develop'. Below this are two tabs: 'Write' (active) and 'Preview'. A rich text editor toolbar is visible with icons for bold, italic, list, code, link, unlink, check, mention, insert, and undo. The text area contains the sentence 'Comentário no arquivo ec2-jenkins.yaml', where 'arquivo' and 'ec2-jenkins.yaml' are underlined in red. At the bottom of the text area, there is a dashed line and the text 'Attach files by dragging & dropping, selecting or pasting them.' with a small icon. A green button labeled 'Create pull request' is located at the bottom right of the form.

Develop

Write Preview

H B I      @  

Comentário no arquivo ec2-jenkins.yaml

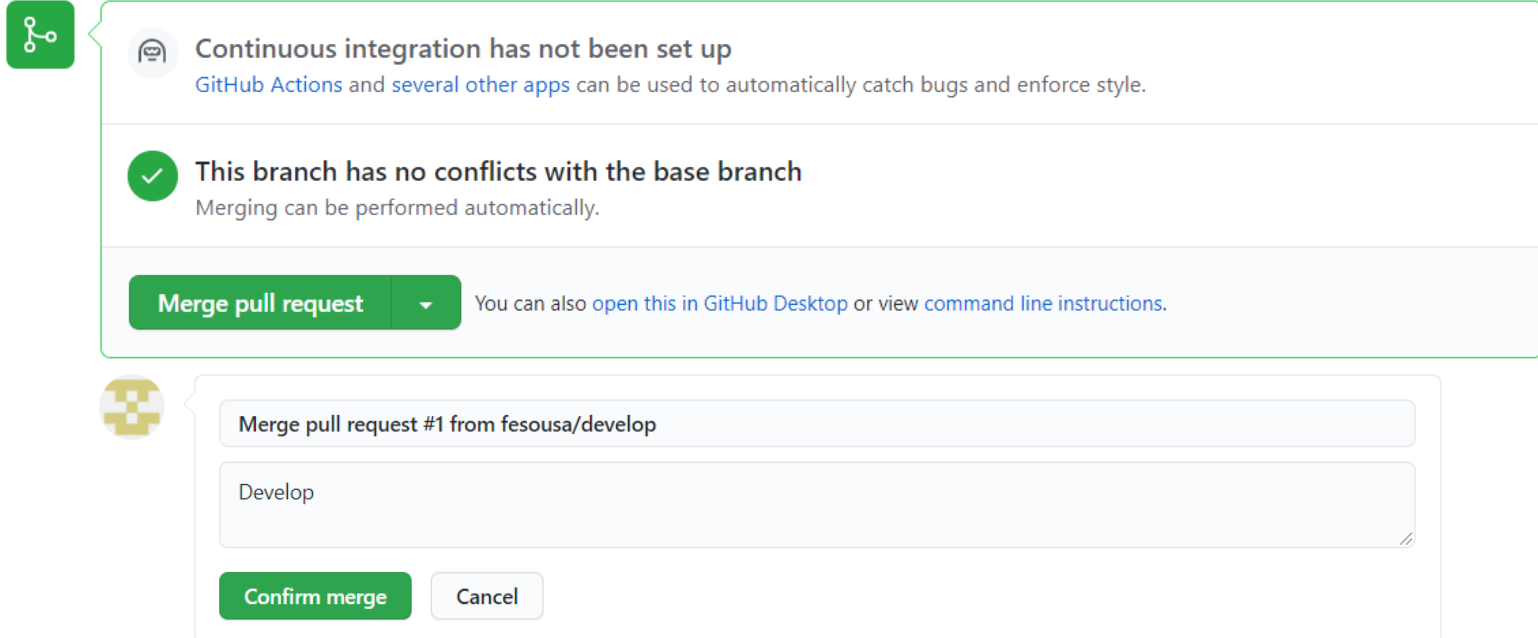
Attach files by dragging & dropping, selecting or pasting them. 

Create pull request

# Pull request

---

- O pull request foi aberto mas a alteração ainda não foi integrada na master
- Na tela no pull request, clique em “merge pull request” e depois em “Confirm merge”




The screenshot displays the GitHub pull request interface. At the top left, there is a green icon with a white branching diagram. Below it, a green box contains the text: "Continuous integration has not been set up" followed by a link: "GitHub Actions and several other apps can be used to automatically catch bugs and enforce style." Below this, a green checkmark icon is next to the text: "This branch has no conflicts with the base branch" followed by the text: "Merging can be performed automatically." Below this, there is a green button labeled "Merge pull request" with a dropdown arrow. To the right of the button, it says: "You can also [open this in GitHub Desktop](#) or view [command line instructions](#)." Below this, there is a yellow and black checkered icon. To its right, there is a text input field containing "Merge pull request #1 from fesousa/develop". Below this, there is another text input field containing "Develop". At the bottom, there are two buttons: a green "Confirm merge" button and a white "Cancel" button.

# Pull request


---


- Volte na branch master, abra o arquivo **ec2-jenkins.yaml** e veja a alteração integrada na master


 master ▾

...

[dataops-lab3](#) / [ec2-jenkins.yaml](#)

 **Frnando Sousa** comentário no arquivo

 History

 0 contributors

1 lines (1 sloc) | 73 Bytes

...

1 # Template CloudFormation para provisionar uma instância EC2 com jenkins

# Boas práticas

---

Faça commits pequenos e de propósito único

Escreva mensagens de commit que indiquem a alteração

Faça commits frequentes

Não altere o histórico

Não envie códigos de construção/compilação e códigos gerados automaticamente

Utilize o arquivo .gitignore para colocar arquivos que não devem ser versionados

# Boas práticas

---

Crie branches para cada alteração que fizer

Não faça push diretamente no ramo principal (master/main/trunk)

Configure a autoria das alterações

Mantenha o repositório local e de trabalho atualizado com a versão mais recente

Não commite trabalhos parciais

Teste bem antes de commitar



# CI/CD

---

- Integração contínua (continuous integration - CI) e entrega contínua (continuous delivery - CD) são duas práticas bem comuns de DevOps
- O objetivo destas práticas é automatizar o processo de integração de novas funcionalidades na versão principal e automatizar o lançamento da nova versão, sempre prezando pela qualidade e velocidade de entrega

# Integração Contínua - CI

---

- A integração contínua visa juntar (merge) em uma branch central as alterações que foram feitas na branch da nova funcionalidade
- Alguns passos são realizados
  - Construção/compilação
  - Testes
  - Integração
- Objetivos:
  - Encontrar e reportar erros rapidamente
  - Aumentar a qualidade
  - Diminuir tempo para validação e lançamento

# Entrega Contínua - CD

---

- A entrega contínua expande o conceito de integração contínua
- Além de construir, testar e integrar as alterações em uma branch central, a prática de CD visa aplicar as alterações em um ambiente
  - Ambiente de testes: entrega contínua (continuous delivery)
    - O pacote já está preparado para produção, mas precisa de aprovação manual
  - Ambiente de produção: implantação contínua (continuous deployment)
    - Todo o processo, desde a integração de branches até a aplicação em produção é automatizado

# CI/CD

---

## CONTINUOUS INTEGRATION

## CONTINUOUS DELIVERY

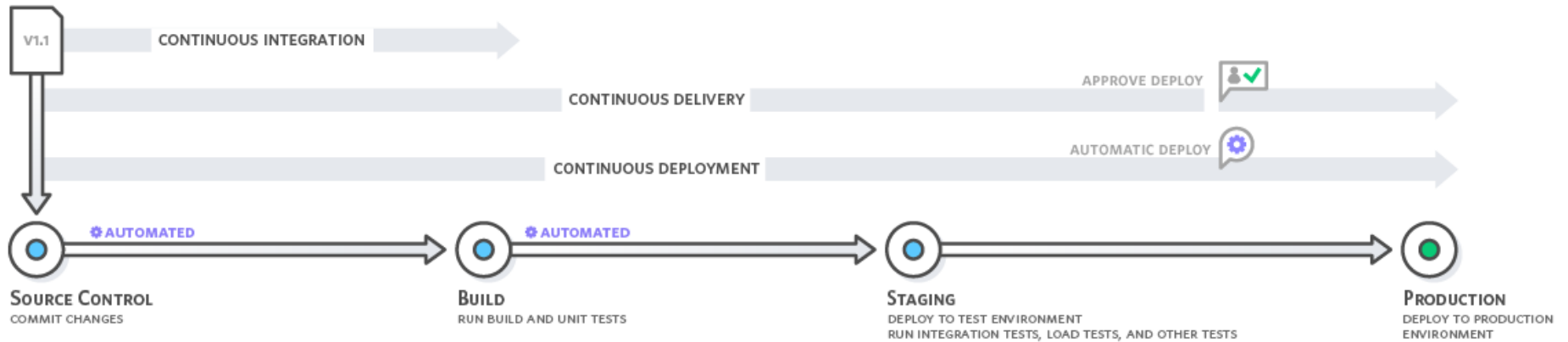
## CONTINUOUS DEPLOYMENT



<https://www.redhat.com/pt-br/topics/devops/what-is-ci-cd>

# CI/CD

---



<https://aws.amazon.com/devops/continuous-delivery/>

# CI/CD + Git

---

- Existem várias soluções de CI/CD disponíveis
  - Jenkins
  - Gitlab CI
  - Bitbucket pipelines
  - Github Actions
  - Azure DevOps Codes
  - Travis CI
  - Circle CI
  - AWS CodeBuild
  - AWS CodePipeline
- Essas soluções podem ser utilizadas separadamente ou em conjunto

# CI/CD + Git

---

- Usualmente as soluções de CI/CD são utilizadas junto de um repositório

- Uma atualização do Código é enviada para uma branch
- A solução de CI/CD detecta a alteração na branch
- O código é testado
- Se os testes forem executados com sucesso, o código é integrado em uma branch principal
- É feita a compilação/construção da aplicação
- Aplicação é colocada em um ambiente de testes
- Testes são realizado
- Se testes passarem, a aplicação pode ser colocada automaticamente em produção

**Continuous integration**

**Continuous delivery**

**Continuous deployment**

# CI/CD + Git – Github Actions

---

- O Github oferece o Github Actions
  - <https://github.com/features/actions>
  - Solução de CI/CD integrada aos repositório do Github
  - Automatização de todo workflow/pipeline da aplicação
    - Compilar/construir
    - Testar
    - Implantar
- Reconhece um evento no repositório, como um push ou um pull request, e executa o workflow automaticamente



# CI/CD + Git – Github Actions

---

- Suporta as principais linguagens do mercado
  - Node.js
  - Python
  - Java
  - .NET
  - Ruby
  - PHP
  - Go
- Suporte a docker
  - Testa a aplicação inteira, com DB
- Workflows pré-definidos
  - Implantar e integrar com as principais ferramentas de mercado

# AWS CodePipeline

---

- AWS CodePipeline é um serviço de entrega contínua (CD) totalmente gerenciado
- Automatiza o pipeline de lançamento para atualizações de aplicações e infraestrutura rápidas e confiáveis
- Automatiza diversas fases do pipeline de lançamento
  - Construção/compilação
  - Testes
  - entrega

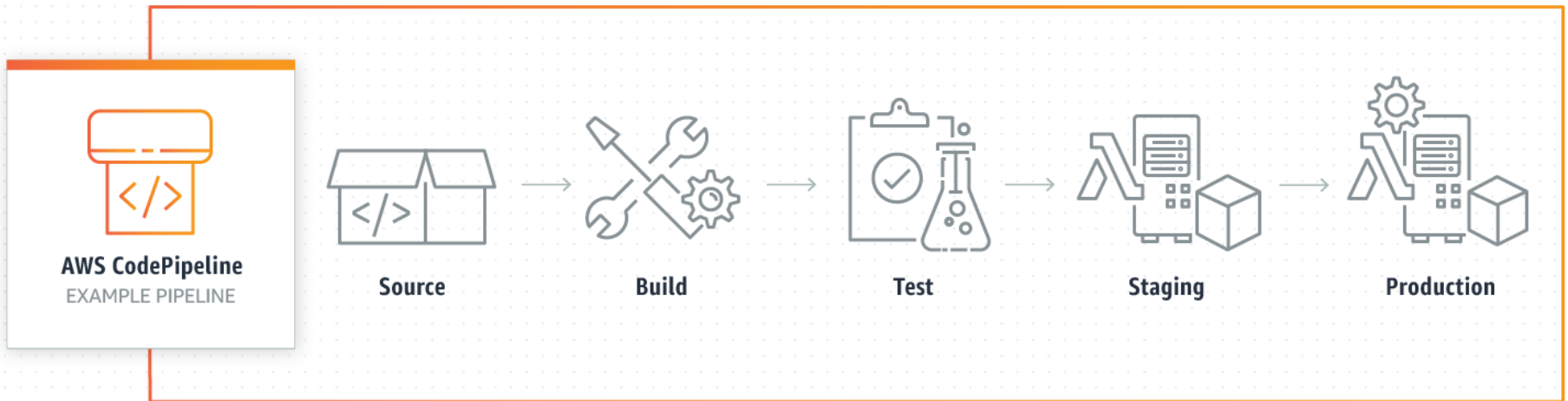
# AWS CodePipeline

---

- O pipeline pode ser executado automaticamente quando detectar uma alteração
  - Diminui interferência manual e chance de erros
- Integra com outros serviços de DevOps
  - GitHub
  - BitBucket
  - Jenkins
  - AWS CodeCommit
  - AWS CodeBuild

# AWS CodePipeline

- Exemplo de um pipeline



[https://aws.amazon.com/codepipeline/?nc1=h\\_ls](https://aws.amazon.com/codepipeline/?nc1=h_ls)

# AWS CodePipeline

---

- Passo 1: Souce (Origem)
- Local onde está o código fonte da aplicação
- Diversas opções de origem
  - AWS CodeCommit (Git da AWS)
  - Amazon ECR (Elastic Container Registry)
  - Amazon S3
  - Bitbucket
  - Github

# AWS CodePipeline

---

- Passo 2: Build (Compilação/Construção)
- Compilação/construção do código fonte
- Opcional, caso defina a próxima etapa de implantação (deploy)
  - É preciso definir pelo menos a etapa de construção ou a implantação
- Nessa etapa que também são realizados os testes
- Opções compilação/construção
  - AWS CodeBuild
  - Jenkins

# AWS CodePipeline

---

- Passo 3: Deploy (Implantação)
- Implantação do código fonte, devidamente construído e testado
- Opcional, caso defina a etapa anterior de construção/compilação (build)
  - É preciso definir pelo menos a etapa de construção ou a implantação

# AWS CodePipeline

---

- Passo 3: Deploy (Implantação)
- Opções de implantação:
  - AWS AppConfig
  - AWS CloudFormation
  - AWS CodeDeploy
  - AWS ElasticBeanstalk
  - AWS OPSWorks Stacks
  - AWS Service Catalog
  - AWS Skills Kit
  - AWS ECS
  - AWS S3



# Jenkins

---

- Jenkins é outra solução de DevOps e CI/CD
- É um serviço de automação de código aberto para construir, testar e implantar as aplicações
- <https://www.jenkins.io/>
- Provê diversos plugins para integração com outros serviços e ferramentas
  - AWS CodePipeline
  - Git
  - Shell Script
  - Python
- Pode substituir as etapas de construção, testes e implantação do CodePipeline

# Lab 3

---

- Veja o documento do laboratório guiado no google classroom
  - DataOps - Lab 3 - Github e CodePipeline
  - Responda ao questionário na atividade

# Obrigado!

Prof. MSc. Fernando Sousa

[fernando.sousa@faculdadeimpacta.com.br](mailto:fernando.sousa@faculdadeimpacta.com.br)