



# WEB ACADEMY

## Integração Contínua

Daniel Augusto Nunes da Silva

# **Apresentação**

# Ementa

- **Entrega de software:** problemas, princípios e *pipelines*. **DevOps**. Controle de versões. **Integração contínua**. Boas práticas no uso de integração contínua. Desenvolvimento Baseado no Trunk (TBD). Servidores de integração contínua. **GitHub Actions**. Fluxo de trabalho no GitHub. **Implantação contínua**. Entrega contínua.



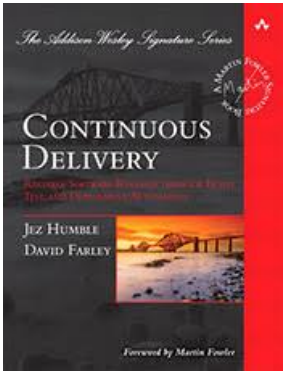
# Objetivos

- **Geral:** Capacitar o aluno na utilização de técnicas de **integração e implantação contínua** em projetos de software, utilizando ferramentas para **automatizar o processo de entregar software**.
- **Específicos:**
  - Discutir os problemas relacionados ao processo de entrega de software;
  - Relacionar os conceitos integração e implantação contínua, com ênfase no conceito de DevOps;
  - Apresentar técnicas e ferramentas para automatizar tarefas relacionadas a integração e implantação contínua.
  - Construir um pipeline de integração e implantação contínua em um projeto de software.

# Conteúdo programático

Introdução

# Bibliografia



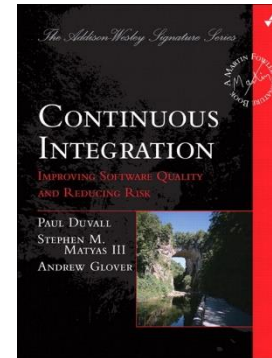
## Continuous Delivery

Jez Humble and David Farley

1ª Edição – 2010

Editora Addison-Wesley

ISBN 9780321601919



## Continuous Integration

Paul Duvall, Steve Matyas e

Andrew Glover

1ª Edição – 2017

Editora Addison-Wesley

ISBN 9780321336385



## Engenharia de Software Moderna

Marco Tulio Valente

<https://engsoftmoderna.info/>



# Sites de referência

- Software Delivery Guide (Martin Fowler).
  - <https://martinfowler.com/delivery.html>
- GitHub Docs: GitHub Actions.
  - <https://docs.github.com/pt/actions>
- Docker Docs.
  - <https://docs.docker.com/get-started/overview/>

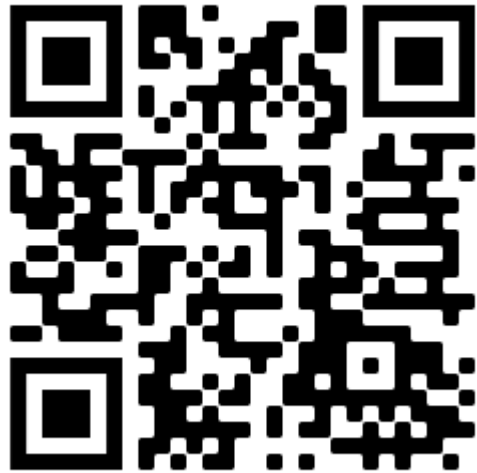
# Ferramentas

- **Railway**

- Plataforma que será utilizada para deploy da aplicação back-end.
- Criar uma conta: <https://railway.app/>



# Contato



<https://linkme.bio/danielnsilva/>

# Introdução

# O problema de entregar software

- **Colocar um software em produção pode implicar em muitas dificuldades:** problemas de compilação, testes, configuração de ambientes, etc.
- **Todo modelo de desenvolvimento de software descreve uma etapa na qual o software entra em operação** e, ainda, recebe novas versões durante seu ciclo de vida.
- Até meados dos anos 2000, normalmente as metodologias estavam concentradas em técnicas de gerenciamento de projetos e requisitos, além de práticas de desenvolvimento e testes.
- Essas boas práticas aplicadas ao processo de desenvolvimento de software devem envolver uma **maneira eficiente de entregar software**.

# Princípios para entrega de software

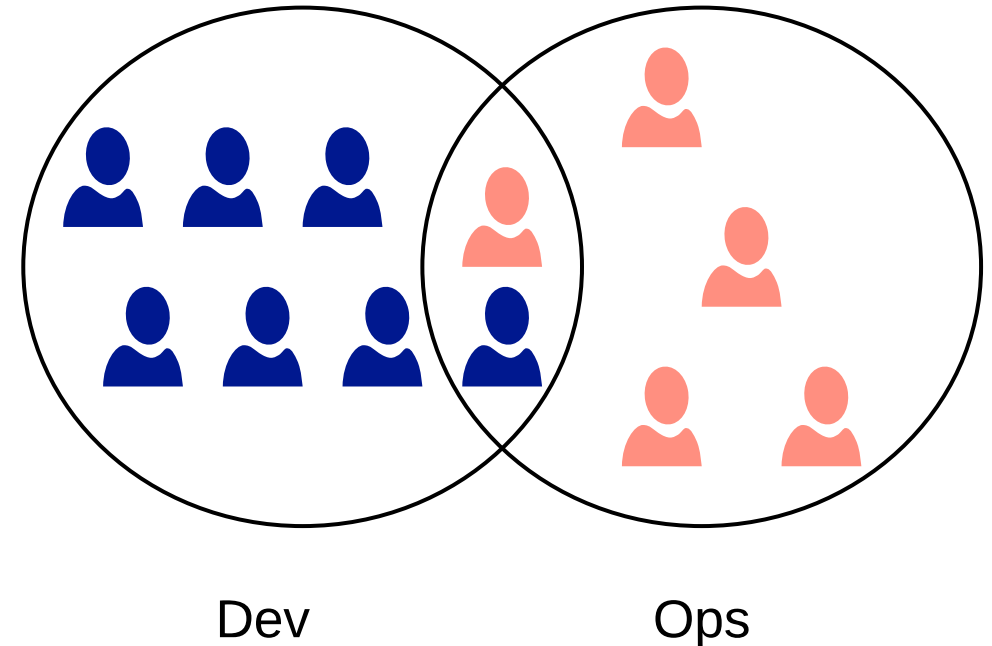
- Crie um **processo repetível e confiável** para entrega de software.
- **Automatize** tudo que for possível.
- Mantenha tudo em um **sistema de controle de versões**.
- Se um passo causa dor, execute-o com **mais frequência** e o quanto antes.
- “**Concluído**” significa pronto para entrega.
- **Todos são responsáveis** pela entrega do software.

# Pipeline para entrega de software



# DevOps

- Historicamente, tarefas de **desenvolvimento** e de **operações** (infraestrutura) são separadas em equipes diferentes.
- Essa divisão dificulta o processo de entregar software com rapidez e qualidade.
- A proposta do **DevOps** é integrar as duas áreas, otimizando o processo de entregar software (colocar em produção).



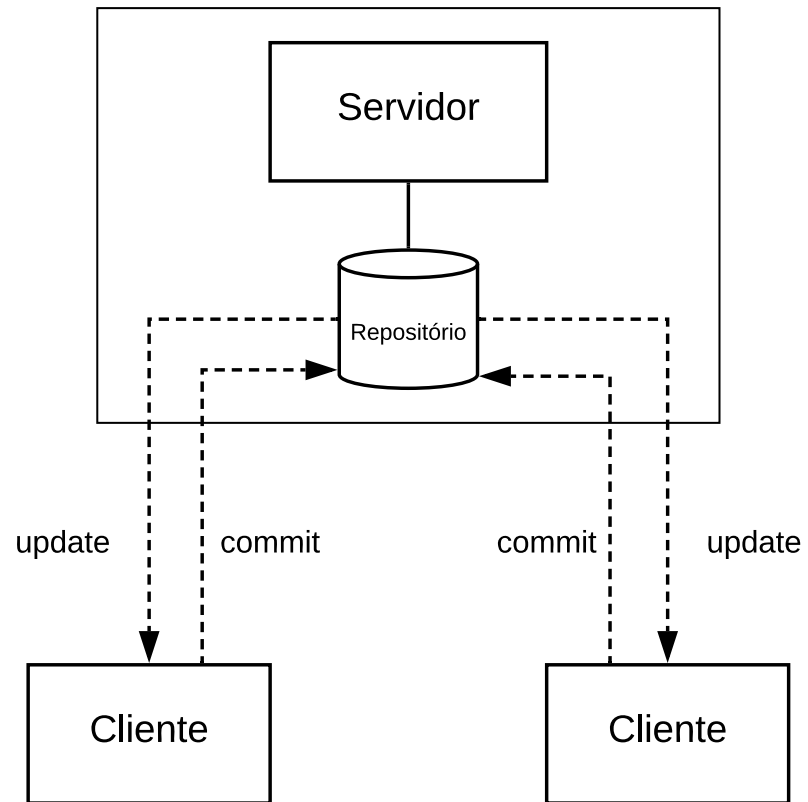
Fonte: VALENTE, 2020.



# Controle de Versões

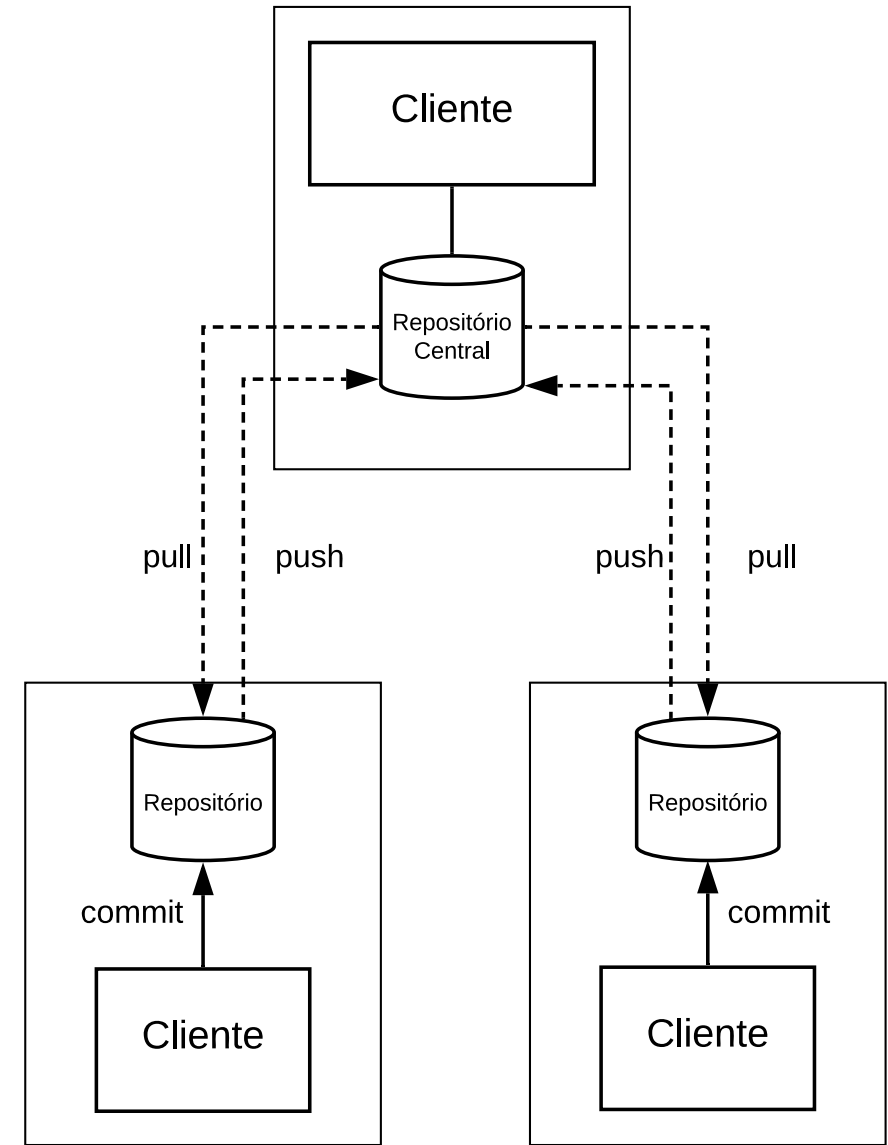
- Um **Sistema de Controle de Versões (VCS)** oferece dois tipos de serviços:
  - Repositório para armazenar as versões mais recentes dos itens de configuração.
  - Permite que se recupere versões mais antigas.
- **VCS** são baseados em uma **arquitetura cliente/servidor**, onde existe um único servidor.
- **Sistemas de Controle de Versões Distribuídos (DVCS)** adota uma **arquitetura peer-to-peer** (descentralizada).

# Controle de Versões



**Centralizado (VCS)**

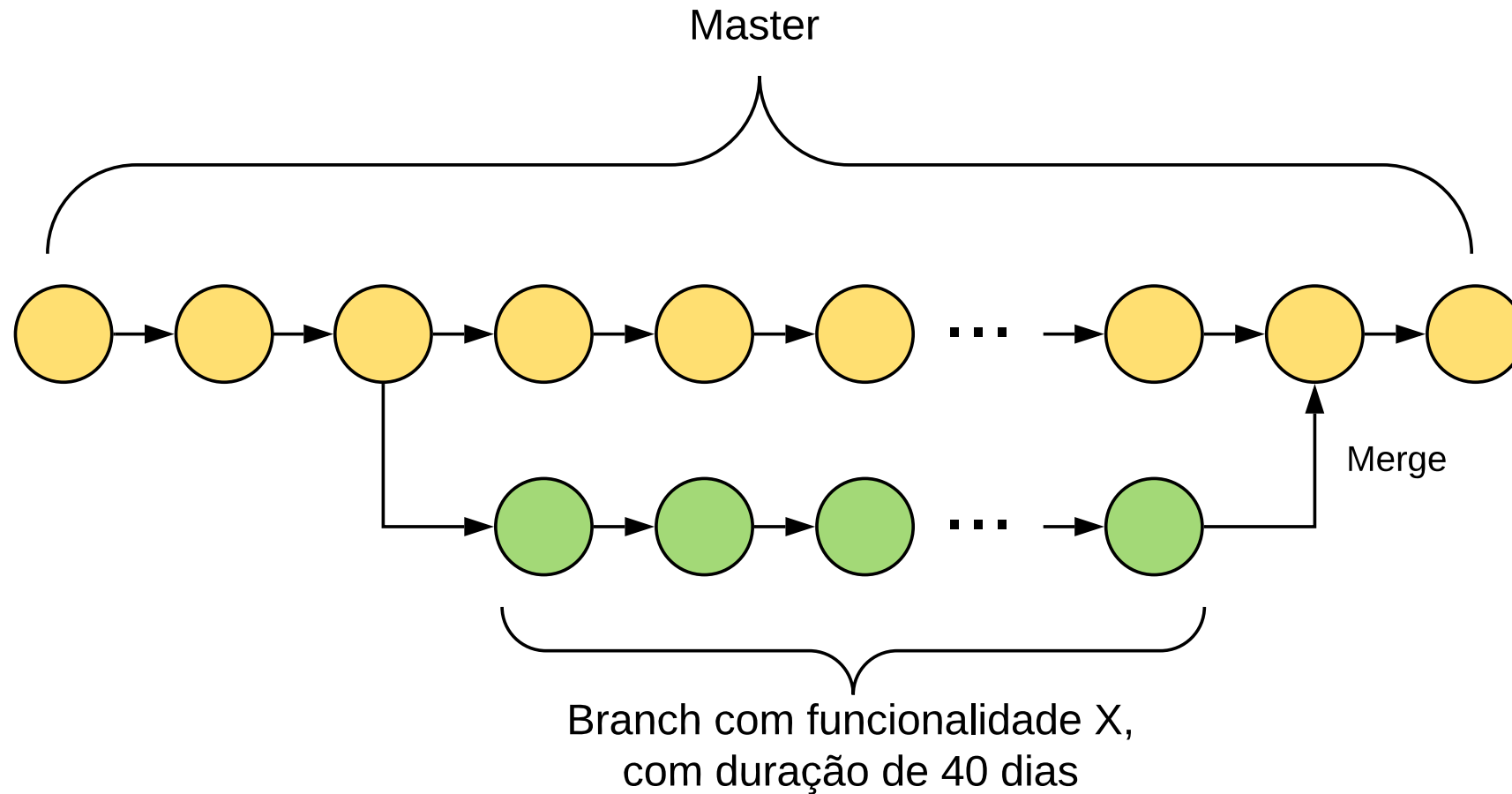
Fonte: VALENTE, 2020.



**Distribuído (DVCS)**

# Integração Contínua

# Integração Contínua



Fonte: VALENTE, 2020.

# Integração Contínua

- **Integração Contínua** (*Continuous Integration* ou **CI**) é uma prática proposta por **Extreme Programming (XP)**.
- Princípio motivador: **se uma tarefa causa dor, não podemos deixar que ela acumule.**
- Devemos **quebrar uma tarefa em subtarefas** que possam ser **realizadas de forma frequente.**

# Integração Contínua

- **Grandes integrações são uma fonte de problemas** para os desenvolvedores, pois eles têm que **resolver de forma manual diversos conflitos**.
- CI recomenda **integrar o código de forma frequente**, isto é, contínua.
- Com isso, as integrações serão pequenas e irão **gerar menos conflitos**.



# Boas Práticas para Uso de CI

- **Build Automatizado**

- É importante que **seja o mais rápido possível**, pois com integração contínua ele será executado com frequência.

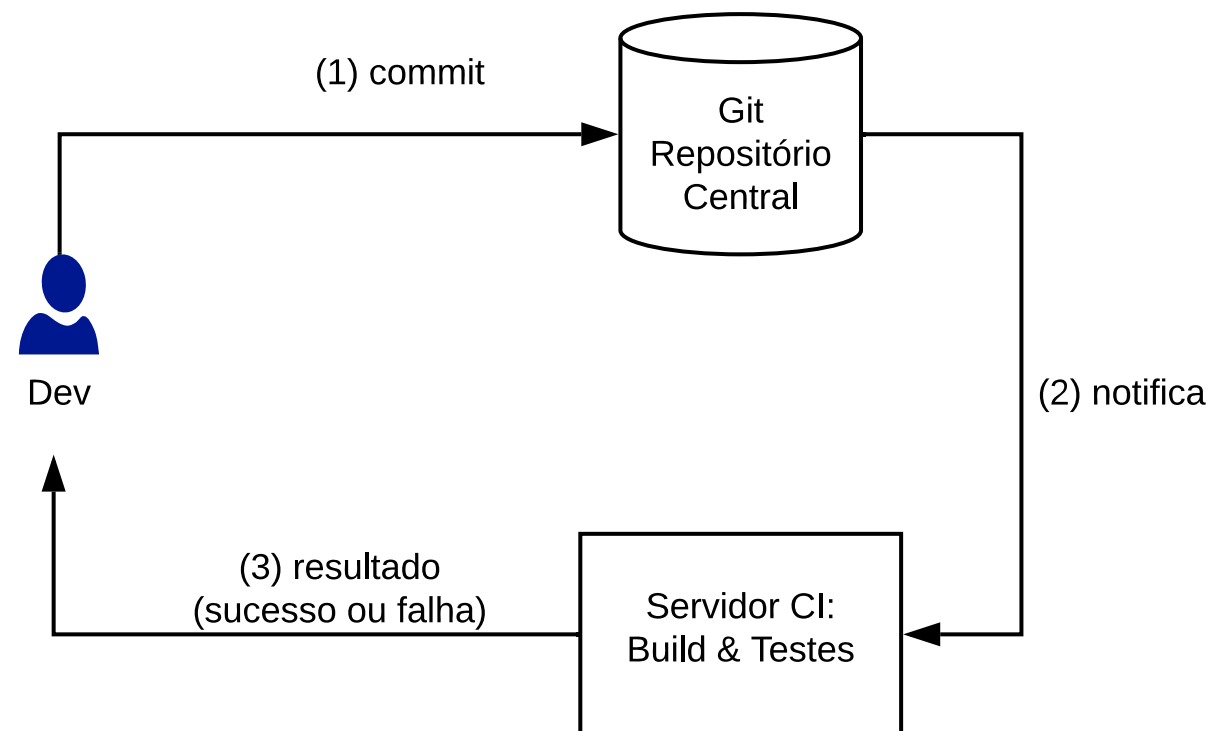
- **Testes Automatizados**

- Além de garantir que o software compila sem erros após cada novo commit, é importante garantir também que ele **continua com o comportamento esperado**.

# Boas Práticas para Uso de CI

## ▪ Servidores de Integração Contínua

- Os builds e testes automatizados devem ser executados com frequência.
- Após um novo commit, o sistema de controle de versões avisa o servidor de CI, que clona o repositório e executa um build completo, bem como roda todos os testes.
- Após a execução do build e dos testes, o servidor notifica o usuário.



Fonte: VALENTE, 2020.

# Boas Práticas para Uso de CI

- **Desenvolvimento Baseado no Trunk**

- CI é compatível com o uso de branches desde que sejam integrados de forma frequente no master (todo dia).
- Quando migram para CI, é comum que as organizações usem também desenvolvimento baseado no trunk (**TBD** – *trunk based development*).
- **Não existem mais branches** para implementação de novas funcionalidades ou para correção de bugs.
- **Todo desenvolvimento ocorre no branch principal**, também conhecido como trunk ou master (main).

# GitHub Actions

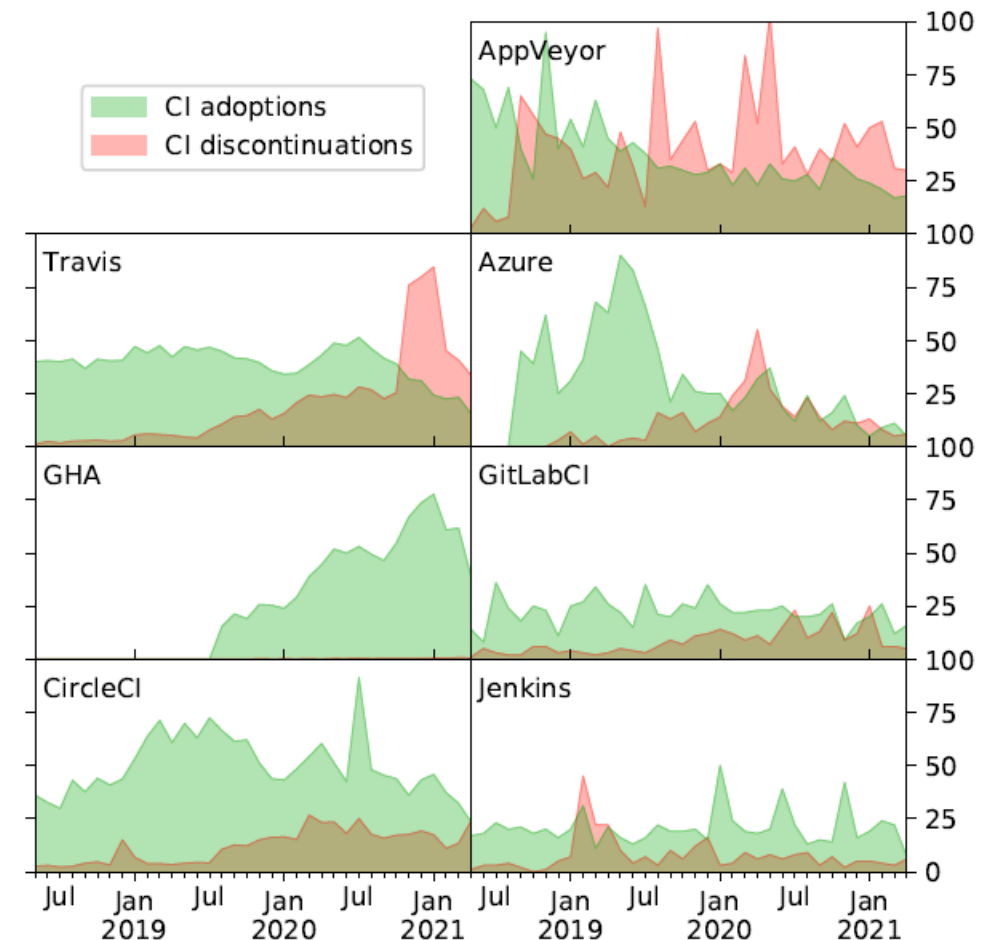
- Ferramenta integrada ao GitHub para automatizar a execução de fluxos de trabalho de desenvolvimento de software, dando suporte a utilização CI.
- **Uma ação (Github Action) é um aplicativo** que executa uma tarefa complexa (envolve vários passos), mas que é repetitiva.
- É possível combinar ações personalizadas e ações já disponíveis na plataforma.
- **O GitHub faz o papel de Servidor de CI**, fornecendo uma máquina virtual para executar as ações.
- O arquivos de configuração utilizam a linguagem YAML e devem ser armazenados no diretório **`.github/workflows/`** em cada projeto.

# Por que GitHub Actions?

## On the rise and fall of of CI services in GitHub

**Abstract**—Continuous integration (CI) services are used in collaborative open source projects to automate parts of the development workflow. Such services have been in widespread use for over a decade, with new CIs being introduced over the years, sometimes overtaking other CIs in popularity. We conducted a longitudinal empirical study over a period of nine years, aiming to better understand this rapidly evolving CI landscape. By analysing the development history of 91,810 GitHub repositories of active npm packages having used at least one CI service, we quantitatively studied the evolution of seven popular CIs, specifically focusing on their co-usage and migration in the considered repositories. We provide statistical evidence of the rise of GitHub Actions, that has become the dominant CI service in less than 18 months time. This coincides with the fall of Travis that has seen an important decrease in usage, likely due to a combination of policy changes and migrations to GitHub Actions.

<http://mehdigolzadeh.com/research/SANER-2022a.pdf>



**Apresentação SANER 2022:**

<https://www.youtube.com/watch?v=TyAIXPAuisE>

# GitHub Actions Workflow

```
1. name: Primeiro Workflow
2. on: [push]
3. jobs:
4.   primeiro-job:
5.     runs-on: ubuntu-latest
6.     steps:
7.     - run: echo "Primeiro passo"
8.     - name: Checkout
9.       uses: actions/checkout@v3
10.    - name: Lista arquivos
11.      run: ls ${github.workspace}
12.    - name: Status
13.      run: echo "Status ${job.status}."
```

**Sempre que um novo arquivo é enviado ao repositório.**

**Provisiona uma máquina virtual para funcionar como Servidor CI.**

**O action "checkout" faz o checkout do branch atual no Servidor CI.**

**O hífen indica um passo, que pode agrupar diferentes configurações.**



# Abordagens para implementar CI

1. **Commit no branch principal:** CI é executado para todos os commits ocorrem no branch principal.
2. **Merge entre branches:** CI é executado quando um commit é feito em algum branch e, ao final do processo, ocorre o merge com o branch principal de forma automática.
3. **Pull request:** CI é executado quando um pull request é aberto, que pode ter origem em um fork do repositório ou um branch. Nesta abordagem também é possível realizar o merge automático com o branch principal.

# GitHub Actions Workflow

```
1. name: CI Workflow
2. on:
3.   push:
4.     branches:
5.       - 'main'
6. jobs:
7.   build:
8.     runs-on: ubuntu-latest
9.     steps:
10.      - uses: actions/checkout@v3
11.      - name: Set up JDK 17
12.        uses: actions/setup-java@v3
13.        with:
14.          java-version: '17'
15.          distribution: 'temurin'
16.      - name: Maven Compile/Test
17.        run: mvn compile test
```

Apenas no branch principal

**Instala a versão 17 do JDK, baseado na distribuição Temurin, no Servidor CI provisionado.**

**Usa o Maven para compilar e testar o projeto.**

# GitHub Actions Workflow

```
1. name: CI Workflow
2. on:
3.   push:
4.     branches:
5.       - '**'
6.       - '!main'
7. jobs:
8.   build:
9.     runs-on: ubuntu-latest
10.    steps:
11.      - uses: actions/checkout@v3
12.        [...]
13.      - name: Merge branch
14.        uses: devmasx/merge-branch@1.4.0
15.        with:
16.          type: now
17.          target_branch: main
18.          github_token: ${{ github.token }}
```

← Todos os branches,  
exceto o principal (main).

Usa o action “merge-branch” para  
fazer merge no branch principal. O  
github.token é gerado  
automaticamente.

# GitHub Actions Workflow

```
1. name: CI Workflow
2. on:
3.   push:
4.     branches:
5.       - 'main'
6.   pull_request:
7. jobs:
8.   build:
9.     runs-on: ubuntu-latest
10.    steps:
11.      - uses: actions/checkout@v3
12.      - name: Set up JDK 17
13.        uses: actions/setup-java@v3
14.        with:
15.          java-version: '17'
16.          distribution: 'temurin'
17.      - name: Maven Compile/Test
18.        run: mvn compile test
```

Quando um Pull Request é criado

Instala a versão 17 do JDK, baseado na distribuição Temurin, no Servidor CI provisionado.

Usa o Maven para compilar e testar o projeto.

# Quando não usar CI?

- **CI tem um limite rígido para integrações no ramo principal:** pelo menos uma integração por dia por desenvolvedor.
  - No entanto, dependendo da organização, do domínio do sistema (que pode ser um sistema crítico) e do perfil dos desenvolvedores (que podem ser iniciantes), pode ser difícil seguir esse limite.
- **CI também não é compatível com projetos de código aberto,** onde os desenvolvedores são voluntários e não têm disponibilidade para trabalhar diariamente no seu código.

# Implantação Contínua



# Implantação Contínua

- Com integração contínua, o código novo é frequentemente integrado no ramo principal, mas **esse código não precisa estar pronto para entrar em produção**.
- Existe mais um passo da cadeia de automação proposta por DevOps, chamado de **Implantação Contínua** (*Continuous Deployment* ou **CD**).
- A diferença entre CI e CD é simples, mas seus impactos são profundos: quando usa-se CD, **todo novo commit** que chega no ramo principal (main) **entra rapidamente em produção** (em questões de horas).

# Implantação Contínua

- **Fluxo de trabalho** quando se usa CD:
  1. O **desenvolvedor cria o novo código e testa** no seu ambiente local.
  2. Ele realiza um commit e o **servidor de CI executa novamente um build e os testes**.
  3. **Algumas vezes no dia, o servidor de CI realiza os testes mais demorados** com os novos commits que ainda não entraram em produção.
  4. **Se todos os testes passarem**, os commits entram imediatamente em produção, e os **usuários já vão interagir com a nova versão do código**.

# Implantação Contínua

- **Vantagens** de CD:
  - CD **reduz o tempo de entrega** de novas funcionalidades.
  - CD torna novas releases (ou implantações) um **não-evento**.
  - Além de reduzir o stress causado por deadlines, CD **ajuda a manter os desenvolvedores motivados**, pois eles não ficam meses trabalhando sem receber feedback.
  - CD **favorece experimentação** e um estilo de desenvolvimento orientado por dados e feedback dos usuários.

# Implantação ou Entrega?

- Implantação Contínua (CD) **não é recomendável para certos tipos de sistemas**, incluindo alguns sistemas desktop, aplicações móveis e aplicações embutidas em hardware.
- É possível usar um versão mais fraca de CD, chamada de **Entrega Contínua** (*Continuous Delivery*): quando se usa entrega contínua, todo commit pode entrar em produção imediatamente, mas **existe uma autoridade externa que toma a decisão sobre quando os commits, de fato, serão liberados para os usuários finais**.
- **Deployment (Implantação)** é o processo de liberar uma nova versão de um sistema para seus usuários.
- **Delivery (Entrega)** é o processo de liberar uma nova versão de um sistema para ser objeto de deployment.

# Implantação com Docker

```
FROM maven:3.8-eclipse-temurin-17-alpine AS build
```

```
COPY src /home/app/src
```

```
COPY pom.xml /home/app
```

```
RUN mvn -f /home/app/pom.xml clean package -Pprod -Dmaven.test.skip=true
```

```
FROM eclipse-temurin:17-jdk-alpine
```

```
COPY --from=build /home/app/target/sgcmapi.jar /home/app/sgcmapi.jar
```

```
EXPOSE 9000
```

```
CMD java -jar /home/app/sgcmapi.jar
```

# GitHub Actions Workflow

```
1.  name: CD Workflow
2.  on: [push]
3.  jobs:
4.    deploy:
5.      runs-on: ubuntu-latest
6.      steps:
7.        - uses: actions/checkout@v3
8.        - uses: actions/setup-node@v3
9.          with:
10.            node-version: 18
11.        - run: npm install -g @railway/cli
12.        - run: railway up --detach
13.      env:
14.        RAILWAY_TOKEN: ${ secrets.RAILWAY_TOKEN }
```

**Instala o Node.js, o cliente da plataforma Railway, e realiza o deploy, informando o token para autenticação.**

# GitHub Actions Workflow

```
1.  name: CD Workflow
2.  on:
3.    schedule:
4.      - cron: '0 23 * * *'
5.  jobs:
6.    deploy:
7.      runs-on: ubuntu-latest
8.      steps:
9.        - uses: actions/checkout@v3
10.       - uses: actions/setup-node@v3
11.         with:
12.           node-version: 18
13.       - run: npm install -g @railway/cli
14.       - run: railway up --detach
15.      env:
16.        RAILWAY_TOKEN: ${ secrets.RAILWAY_TOKEN }
```

# GitHub Actions Workflow

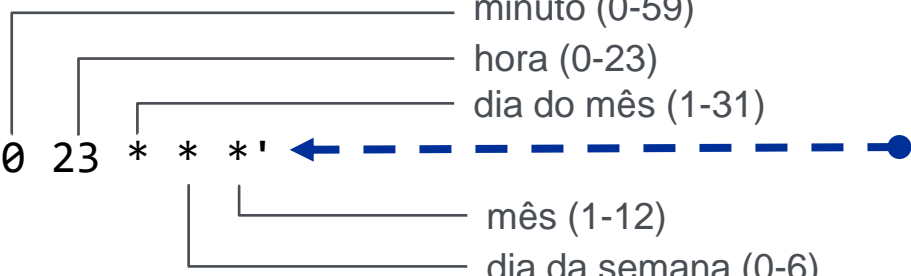
```
1.  name: CD Workflow
2.  on:
3.    schedule:
4.      - cron: '0 23 * * *'
5.  jobs:
6.    deploy:
7.      runs-on: ubuntu-latest
8.      steps:
9.        - uses: actions/checkout@v3
10.       - uses: actions/setup-node@v3
11.         with:
12.           node-version: 16
13.       - run: npm install -g @railway/cli
14.       - run: railway up --detach
15.     env:
16.       RAILWAY_TOKEN: ${ secrets.RAILWAY_TOKEN }
```

**Executa o workflow todos os dias às 23:00h**



# GitHub Actions Workflow

```
1. name: CD Workflow
2. on:
3.   schedule:
4.     - cron: '0 23 * * *'
5. jobs:
6.   deploy:
7.     runs-on: ubuntu-latest
8.     steps:
9.       - uses: actions/checkout@v3
10.      - uses: actions/setup-node@v3
11.        with:
12.          node-version: 16
13.      - run: npm install -g @railway/cli
14.      - run: railway up --detach
15.     env:
16.       RAILWAY_TOKEN: ${ secrets.RAILWAY_TOKEN }
```



**Executa o workflow todos os dias às 23:00h**

**Fim!**



# Referências

- HUMBLE, Jez; FARLEY, David. **Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation**. 1. ed. [S. l.]: Pearson Addison-Wesley, 2010. 512 p.
- DUVALL, Paul M. **Continuous Integration: Improving Software Quality and Reducing Risk**. 1. ed. [S. l.]: Pearson Addison-Wesley, 2007. 336 p.
- GITHUB (ed.). **GitHub Docs: GitHub Actions**. [S. l.], 2023. Disponível em: <https://docs.github.com/pt/actions>.
- MARCO TULIO VALENTE. **Engenharia de Software Moderna: Princípios e Práticas para Desenvolvimento de Software com Produtividade**, 2020. Disponível em: <https://engsoftmoderna.info/>.
- SOMMERVILLE, Ian. **Engenharia de Software**. 9. ed. São Paulo: Pearson Addison-Wesley, 2011.