

Passo-a-passo detalhado para configurar um pipeline de integração contínua com execução manual, verificação de formatação, execução de testes E2E com Playwright e relatório de testes, tanto no **Azure DevOps** quanto no **GitHub Actions**.

✓ Parte 1 – GitHub Actions

🔧 Objetivo:

Configurar um workflow de execução manual que:

- Instale dependências com Yarn
 - Verifique formatação com Prettier
 - Execute testes E2E com Playwright
 - Gere relatório de testes com [dorny/test-reporter](#)
-

📄 Arquivo [.github/workflows/01-manual-exec.yaml](#):

```
# CI de Nível 01 - Disparo manual a partir de 1 clique

# nome do nosso 'pipeline'
name: 'Execução Manual'

# regras de disparo (gatilhos/triggers)
# gatilho de exec manual = workflow_dispatch
on:
  workflow_dispatch:

# trabalhos/tarefas dentro do pipeline = jobs
jobs:
  e2e-tests:
    runs-on: ubuntu-latest
```

```
steps:
  # clone do projeto
  - uses: actions/checkout@v4
    with:
      submodules: false
      clean: true
      fetch-depth: 0

  # instalação do Node.js
  - uses: actions/setup-node@v4
    with:
      node-version: 22.x

  # instalação do Yarn
  - name: Instalando Yarn
    run: npm install -g yarn

  # instalação das dependências
  - name: Instalando dependências
    run: yarn

  # Verificação de formatação de código com Prettier
  - name: Checando formatação com Prettier
    run: yarn run format:check

  # instalação do Playwright
  - name: Instalando Playwright
    run: yarn playwright install

  # execução dos testes
  - name: Executando testes E2E
    run: yarn run e2e

  # Gera o relatório após os testes
  - name: Publicar resultado no PR
    uses: dorny/test-reporter@v1
    if: ${{ !cancelled() }}
    with:
      name: Test Report
      path: ./test-results/results.xml
      reporter: java-junit
```

Dicas úteis:

- Certifique-se que o comando `yarn run e2e` realmente gera `results.xml`.
- O erro `exit code 128` pode ser causado por diretórios inseguros no Git – adicione o repositório como trusted:

```
git config --global --add safe.directory C:/path/do/repositorio
```

Parte 2 – Azure DevOps Pipelines

Pré-requisitos:

- Projeto criado no Azure DevOps
- Código hospedado em repositório Git (Azure ou GitHub)
- Acesso a Pipelines (YAML)

Etapas de configuração:

1. Criação do pipeline

- Acesse **Pipelines > Pipelines** no Azure DevOps
 - Clique em "**New Pipeline**"
 - Escolha **repositório Git** (Azure ou GitHub)
 - Escolha a opção **YAML**
 - Dê um nome e salve o arquivo `azure-pipelines.yml`
-

Exemplo de **azure-pipelines.yml**

```
name: 'Execução Manual - $(Date:yyyyMMdd)$(Rev:r) '

trigger:
  - main

pool:
  name: Default

steps:
  - checkout: self
    displayName: 'Clonar o repositório'

  - task: NodeTool@0
    inputs:
      versionSpec: '22.x'
    displayName: 'Instalar Node.js'

  - script: npm install -g yarn
    displayName: 'Instalar Yarn'

  - script: yarn
    displayName: 'Instalar dependências'

  - script: yarn run format:check
    displayName: 'Checando Formatação com Prettier'

  - script: yarn playwright install
    displayName: 'Instalar Playwright'

  - script: yarn run e2e
    displayName: 'Executar testes E2E'

  - task: PublishTestResults@2
    displayName: 'Publicar Relatório de Testes'
    inputs:
      testResultsFormat: 'JUnit'
      testResultsFiles: '**/results.xml'
    condition: always()
```

Execução manual:

- Após configurar o YAML, vá em **Pipelines**
- Clique no pipeline criado
- Clique em **"Run pipeline"** para execução manual

Extras:

- Se quiser adicionar gatilhos automáticos, basta alterar `trigger` ou `pr`.
- Para gerar relatório HTML, pode-se usar Playwright com `html-report`, mas será necessário publicar como artifact com `PublishBuildArtifacts@1`.

◆ 3. Configurando Self-Hosted Runner (Máquina Local)

Pré-requisitos

- Windows, Linux ou macOS com acesso à internet.
- Node.js, Yarn e Git instalados.

Passos para GitHub

3.1 Acessar configurações do repositório:

- Vá até **Settings > Actions > Runners**.
- Clique em **"New self-hosted runner"**.
- Selecione o SO e siga os comandos:

3.2 Exemplo para Windows:

```
mkdir actions-runner && cd actions-runner

Invoke-WebRequest -Uri
https://github.com/actions/runner/releases/download/v2.315.0/actions-runner-win-x64-2.315.0.zip -OutFile actions-runner.zip

Expand-Archive -Path actions-runner.zip

.\config.cmd --url https://github.com/SEU_USUARIO/SEU_REPOSITORIO --token
SEU_TOKEN

.\run.cmd
```

⚠ A máquina deve permanecer ligada e o script `run.cmd` ativo.

Passos para Azure DevOps

3.1 Acesse:

Project Settings > Agent pools > Default > New Agent

3.2 Baixe e configure o agente:

```
mkdir myagent && cd myagent

Invoke-WebRequest -Uri
https://vstsagentpackage.azureedge.net/agent/3.233.3/vsts-agent-win-x64-3.233.3.zip -OutFile agent.zip

Expand-Archive -Path agent.zip

.\config.cmd
# Insira:
# > URL do GitHub
# > Token PAT (com escopo "Agent Pools (read, manage)")

.\run.cmd
```

🔑 Como gerar o Personal Access Token (PAT) no GitHub

🔧 Passo a Passo detalhado:

1. Acesse sua conta no GitHub

- Vá para: <https://github.com>

2. Acesse as configurações

- No canto superior direito, clique na sua **foto de perfil** > selecione **"Settings"**.

3. Vá até Developer settings

- Role até o final do menu lateral esquerdo.
- Clique em **"Developer settings"**.

4. Acesse Personal access tokens

- No menu à esquerda, clique em **"Personal access tokens"** > **"Tokens (classic)"** (ou **"Fine-grained tokens"** se preferir granularidade extra).

- Clique em **"Generate new"**

6. Clique em "Generate token"

7. Copie o token imediatamente

⚠️ Você só verá o token **UMA VEZ!**
Copie e salve com segurança (ex: gerenciador de senhas).

🔧 Onde usar esse token?

Você vai utilizá-lo ao **configurar o runner self-hosted** com o comando:




```
./config.sh  
# ou no Windows:  
config.cmd
```

Durante a configuração, o terminal vai solicitar:

- **URL do repositório:** <https://github.com/sua-org/seu-repo>
 - **Tipo de autenticação:** selecione **token**
 - **Cole o token PAT que você gerou**
-

Conclusão

Você configurou:

-  Pipeline manual no **Azure DevOps** com relatórios e Playwright.
-  Workflow no **GitHub Actions** com formatação e E2E tests.
-  **Runner local** (self-hosted) para ambos os sistemas.