PIPELINE CI/CD - GITHUB ACTIONS

CURIOSIDADES:

A tradução de YAML é "YAML Ain't Markup Language", que em português seria "YAML Não é Linguagem de Marcação". Esta é uma sigla recursiva (um tipo de piada de programador) que enfatiza que YAML não é uma linguagem de marcação como XML ou HTML. O propósito de YAML é ser uma linguagem de serialização de dados que seja fácil de ler e escrever por humanos, com foco em simplicidade e clareza.

YAML (YAML Ain't Markup Language) é um formato de serialização de dados legível por humanos que é comumente usado para configurar arquivos, definir parâmetros e armazenar dados de maneira estruturada. Ele é especialmente popular em contextos de DevOps e CI/CD devido à sua simplicidade e clareza. Aqui estão alguns pontos importantes sobre YAML:

Características Principais

1. **Legibilidade**: YAML é projetado para ser fácil de ler e escrever por humanos. A estrutura de indentação torna a hierarquia dos dados clara.
2. **Formato Simples**: Usa indentação (espaços) para representar a estrutura, em vez de caracteres especiais como chaves ou colchetes.
3. **Flexível**: Pode representar uma ampla variedade de tipos de dados, incluindo listas, mapas (dicionários), strings, números, booleanos e nulos.
4. **Sem Marcadores**: Diferente do XML e JSON, YAML não usa marcadores como `<tag>` ou `{}` para estruturar os dados.

Estrutura e Sintaxe

Aqui estão alguns exemplos de estruturas e sintaxes comuns em YAML:

Mapas (Dicionários)

```yaml
database:
  host: localhost
  port: 5432
  username: user
  password: pass
```

Listas

```yaml
services:
  - web
  - database
  - cache
```

Aninhamento de Dados

```yaml
```

```yaml
stages:
  - name: Build
    steps:
      - name: Checkout code
        uses: actions/checkout@v2
      - name: Set up Node.js
        uses: actions/setup-node@v2
        with:
          node-version: '14'
```

Comentários

```yaml
# Isto é um comentário
environment: production
```

Uso em Workflows do GitHub Actions

No contexto do GitHub Actions, arquivos YAML são usados para definir workflows de CI/CD. Aqui está um exemplo básico:

```yaml
name: CI Workflow

on:
  push:
    branches:
      - main
  workflow_dispatch:

jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout repository
        uses: actions/checkout@v2

      - name: Set up Node.js
        uses: actions/setup-node@v2
        with:
          node-version: '14'

      - name: Install dependencies
        run: npm install

      - name: Run tests
        run: npm test
```

Vantagens do YAML

1. Simplicidade: A ausência de sintaxe complexa torna o YAML fácil de ler e escrever.
2. Hierarquia Clara: A indentação representa claramente a hierarquia dos dados.
3. Integração com Ferramentas: YAML é suportado por muitas ferramentas de automação, como Ansible, Kubernetes, e GitHub Actions.

Desvantagens do YAML

1. Sensível a Espaços: YAML depende fortemente da indentação, o que pode levar a erros difíceis de detectar se a indentação não for consistente.
2. Complexidade com Dados Aninhados: Pode se tornar difícil de ler em casos de estruturas de dados altamente aninhadas.

Comparação com Outros Formatos

- JSON: Também legível por humanos e comumente usado em APIs. JSON é mais rígido na sintaxe (uso de chaves e colchetes) e não permite comentários.
- XML: Mais verboso e usa tags de abertura e fechamento, tornando-o mais difícil de ler e escrever para humanos. No entanto, XML é mais adequado para documentos complexos e dados com esquemas rigorosos.

Em resumo, YAML é uma escolha popular para arquivos de configuração devido à sua simplicidade e legibilidade, sendo amplamente utilizado em pipelines de CI/CD, incluindo workflows de GitHub Actions.

PRÁTICA GitHub Actions:

Vamos apresentar ações via workflow. Esta ação pode, por exemplo, **verificar a sintaxe HTML** do seu site antes de fazer o deploy. **Vamos utilizar a ferramenta [HTMLHint](https://htmlhint.com/)** para isso.

Aqui está um workflow que inclui uma etapa para verificar a sintaxe HTML e uma etapa para fazer o deploy para o GitHub Pages:

```
name: Check HTML and Deploy to GitHub Pages

on:
  push:
    branches: ["main"]
  workflow_dispatch:

permissions:
  contents: read
  pages: write
  id-token: write

concurrency:
  group: "pages"
  cancel-in-progress: false
```

```yaml
jobs:
  lint:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout
        uses: actions/checkout@v4

      - name: Setup Node.js
        uses: actions/setup-node@v3
        with:
          node-version: '16'

      - name: Install HTMLHint
        run: npm install -g htmlhint

      - name: Run HTMLHint
        run: htmlhint **/*.html

  deploy:
    needs: lint
    environment:
      name: github-pages
      url: ${{ steps.deployment.outputs.page_url }}
    runs-on: ubuntu-latest
    steps:
      - name: Checkout
        uses: actions/checkout@v4

      - name: Setup Pages
        uses: actions/configure-pages@v5

      - name: Upload artifact
        uses: actions/upload-pages-artifact@v1
        with:
          path: .

      - name: Deploy to GitHub Pages
        id: deployment
        uses: actions/deploy-pages@v1
```

Explicação das Etapas:

1. lint Job:
- Checkout: Faz o checkout do código-fonte.
- Setup Node.js: Configura o Node.js para a execução de scripts.
- Install HTMLHint: Instala o HTMLHint globalmente usando npm.
- Run HTMLHint: Executa o HTMLHint para verificar a sintaxe de todos os arquivos HTML no repositório.

2. deploy Job:
- needs: lint: Indica que o job `deploy` só será executado se o job `lint` for bem-sucedido.

- Checkout: Faz o checkout do código-fonte.
- Setup Pages: Configura o GitHub Pages.
- Upload artifact: Carrega os artefatos (arquivos HTML e outros recursos) para serem publicados.
- Deploy to GitHub Pages: Faz o deploy dos artefatos para o GitHub Pages.

No contexto dos workflows do GitHub Actions, o Node.js é executado em máquinas virtuais hospedadas na infraestrutura do GitHub. Essas máquinas virtuais são configuradas com diferentes sistemas operacionais (como Ubuntu, Windows ou macOS) dependendo das necessidades do seu workflow e das especificações definidas no arquivo YAML do workflow.

Quando você utiliza a ação `actions/setup-node`, por exemplo, você especifica a versão do Node.js que deseja utilizar (`node-version`). O GitHub Actions então provisiona uma máquina virtual com o sistema operacional correspondente e a versão do Node.js solicitada. A execução do seu script Node.js, ou qualquer outro tipo de script ou comando que você configure no seu workflow, ocorre dentro dessa máquina virtual.

É importante lembrar que o ambiente de execução do GitHub Actions é isolado para cada execução de workflow, garantindo que as dependências e configurações especificadas no seu workflow sejam consistentemente aplicadas durante a execução.

Passo a Passo:

1. Adicionar o Arquivo de Workflow:
- No repositório, navegue até a pasta `.github/workflows/`.
- Dentro da pasta `workflows`, crie um novo arquivo chamado `deploy.yml` (ou qualquer outro nome que você prefira) e cole o código acima.

2. Adicionar o Arquivo `index.html`:
- No diretório raiz do seu repositório, crie um arquivo chamado `index.html` e adicione o conteúdo HTML desejado.

3. Comitar e Empurrar as Mudanças:
- Comite o arquivo de workflow e o conteúdo estático no branch `main`.
- Empurre as mudanças para o repositório no GitHub.

4. Executar o Workflow:
- Navegue até a aba "Actions" no repositório do GitHub.
- Você deve ver o workflow que você criou listado. Clique nele para ver os detalhes e certifique-se de que ele foi acionado corretamente.
- Você também pode iniciar o workflow manualmente clicando no botão "Run workflow" se necessário.

5. Verificar a Página no GitHub Pages:
- Após a execução bem-sucedida do workflow, vá para a aba "Settings" do repositório.
- No menu lateral, clique em "Pages".
- Lá você verá a URL do site do GitHub Pages onde seu conteúdo estático foi publicado.

Com esses passos, você terá um workflow que verifica a sintaxe HTML e, se estiver tudo certo, faz o deploy do conteúdo para o GitHub Pages.

Os exemplos de workflows que discutimos podem ser considerados pipelines de CI/CD (Integração

Contínua e Entrega/Implantação Contínua). Vamos detalhar como esses exemplos se encaixam nos conceitos de CI/CD:

1. Integração Contínua (CI):
- A integração contínua envolve a automação do processo de construção e teste do código sempre que mudanças são introduzidas no repositório.
- No exemplo discutido, a etapa de "lint" usando o `HTMLHint` verifica a sintaxe HTML do projeto sempre que há um push para o branch `main`. Este é um exemplo clássico de uma prática de CI, onde o código é continuamente integrado e verificado.

2. Entrega/Implantação Contínua (CD):
- A entrega contínua refere-se à prática de preparar continuamente o código para um lançamento para produção.
- A implantação contínua vai um passo além, automatizando o processo de implantação para produção.
- No nosso exemplo, após a verificação de sintaxe HTML bem-sucedida, o workflow continua com as etapas de implantação, onde o conteúdo do site é implantado automaticamente no GitHub Pages. Isto exemplifica a prática de CD, automatizando o processo desde a verificação até a implantação final.

Exemplos de Workflows CI/CD

Pipeline CI:

```
name: Check HTML

on:
  push:
    branches: ["main"]
  pull_request:
    branches: ["main"]

jobs:
  lint:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout
        uses: actions/checkout@v4

      - name: Setup Node.js
        uses: actions/setup-node@v3
        with:
          node-version: '16'

      - name: Install HTMLHint
        run: npm install -g htmlhint

      - name: Run HTMLHint
        run: htmlhint **/*.html
```

Pipeline CI/CD:

```
name: Check HTML and Deploy to GitHub Pages
```

```yaml
on:
  push:
    branches: ["main"]
  workflow_dispatch:

permissions:
  contents: read
  pages: write
  id-token: write

concurrency:
  group: "pages"
  cancel-in-progress: false

jobs:
  lint:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout
        uses: actions/checkout@v4

      - name: Setup Node.js
        uses: actions/setup-node@v3
        with:
          node-version: '16'

      - name: Install HTMLHint
        run: npm install -g htmlhint

      - name: Run HTMLHint
        run: htmlhint **/*.html

  deploy:
    needs: lint
    environment:
      name: github-pages
      url: ${{ steps.deployment.outputs.page_url }}
    runs-on: ubuntu-latest
    steps:
      - name: Checkout
        uses: actions/checkout@v4

      - name: Setup Pages
        uses: actions/configure-pages@v5

      - name: Upload artifact
        uses: actions/upload-pages-artifact@v1
        with:
          path: .
```

```
    - name: Deploy to GitHub Pages
      id: deployment
      uses: actions/deploy-pages@v1
```

Resumo

- **CI:** Envolve a automação da construção e teste do código. No exemplo, a verificação de sintaxe HTML é parte da CI.
- **CD:** Envolve a automação da entrega e implantação do código. No exemplo, a implantação para o GitHub Pages é parte da CD.

Com esses workflows, você está utilizando práticas de CI/CD para garantir que o código seja continuamente integrado e entregue/implantado de forma automatizada e eficiente.