DevOps

UNIVERSIDADE DO VALE DO SAPUCAÍ – UNIVAS – POUSO ALEGRE/MG
PROFESSOR: RAFFAEL CARVALHO

Entrega Contínua **(CD)**

Da Integração Contínua (CI) à Implantação Contínua (CD)

- Na última aula, configuramos o CI: rodamos testes e, após o merge na main, publicamos nossa imagem Docker no **GitHub Container Registry (GHCR)**. Isso é "Entrega Contínua".
- Nessa aula, vamos adicionar a **Implantação Contínua (CD)**: vamos fazer com que, após a publicação, o GitHub Actions **implante automaticamente** nossa aplicação em um "servidor". Para esta aula, nosso "servidor" será a **nossa própria máquina local**.

Ferramentas Necessárias

- O repositório da aula anterior.
- Docker Desktop (precisa estar instalado e rodando).
- Um GitHub Self-hosted Runner (o "agente" que conecta o GitHub à sua máquina).
- Um arquivo docker-compose.yml (que vamos criar).

Ponto de Partida (O Pipeline de CI da Última Aula)

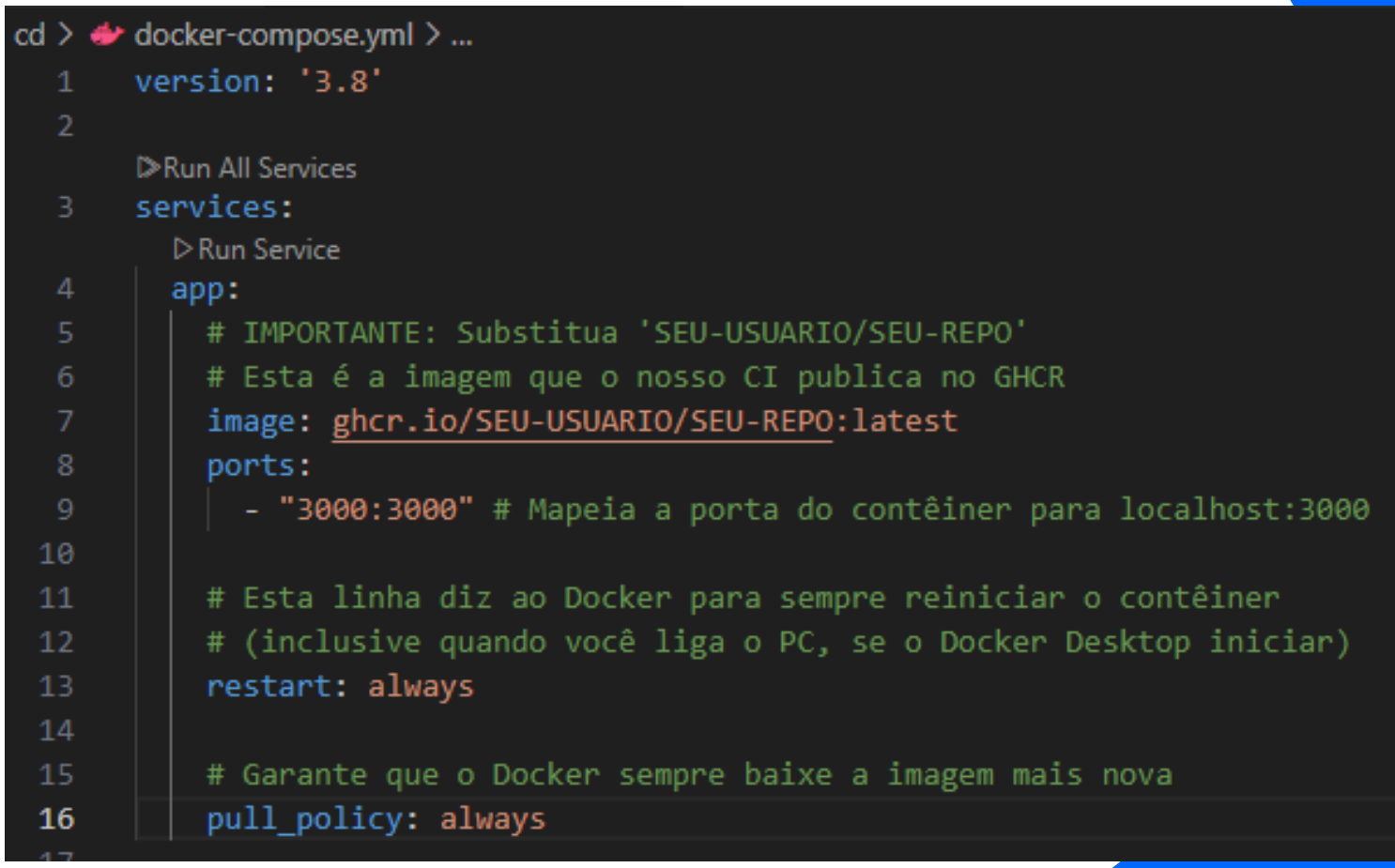
Esta fase é apenas uma revisão. O seu arquivo .github/workflows/ci.yml (do final da aula anterior) já deve estar assim:

- **Gatilhos:** Roda em push e pull_request para a branch main.
- **Serviço (test-and-build):**
  1. Faz Checkout do código.
  2. Configura o Node.js.
  3. Roda npm install.
  4. Roda npm test.
  5. **Se for um push na main**: Loga no GHCR, constrói a imagem Docker e faz **push** dela para o GHCR.
- Agora, vamos adicionar um **segundo serviço** para fazer o deploy.

Preparando o "Servidor" (Sua Máquina Local)

Precisamos de uma forma de rodar a imagem que publicamos no GHCR. Para isso, usaremos o docker-compose.

1. **Crie o docker-compose.yml**: Na raiz do seu projeto ola-devops, crie um novo arquivo chamado docker-compose.yml.



Conectando o GitHub ao "Servidor" (Self-hosted Runner)

Este é o "agente" que permite ao GitHub Actions executar comandos na sua máquina.

1. **Atualize o .gitignore:** O Runner instala muitos arquivos. Não queremos enviá-los ao repositório. Abra seu arquivo .gitignore e adicione esta linha no final: `/actions-runner`
2. Instale o Runner:
   - No seu repositório do GitHub, vá em Settings > Actions > Runners.
   - Clique em "**New self-hosted runner**".
   - Siga as instruções de "Download" e "Configure". Execute esses comandos de dentro da pasta do seu projeto. Isso criará a pasta actions-runner (que agora está no .gitignore).
3. Inicie o Runner:
   1. Após a configuração, inicie o agente. Ele precisa ficar rodando para "ouvir" por jobs.

      ```
      # (Dentro da pasta actions-runner)
      ./run.sh
      # (ou run.cmd no Windows)
      ```
   2. Deixe este terminal aberto!
   3. Você deve ver a mensagem: **"Listening for jobs..."**

Atualizando o Pipeline para CD (ci.yml)

Agora, vamos adicionar o **segundo serviço (deploy-to-local)** ao nosso pipeline. Este novo serviço irá rodar *depois* do serviço da última aula (test-and-build) e fará o deploy. Adicione o texto ao lado ao final do arquivo ci.yml.

```yaml
  deploy-to-local:
    # Este job SÓ RODA se for um push na 'main'
    if: github.event_name == 'push' && github.ref == 'refs/heads/main'

    # CORREÇÃO: Ele precisa que o Job 1 (test-and-build) termine com sucesso
    needs: test-and-build

    # Ele roda no seu PC (onde o runner está ouvindo)
    runs-on: self-hosted

    steps:
      # 1. Pega o código (para ter o docker-compose.yml)
      - name: Checkout do Código
        uses: actions/checkout@v4

      # 2. Loga no GHCR (para poder BAIXAR a imagem)
      - name: Logar no GitHub Container Registry (GHCR)
        uses: docker/login-action@v3
        with:
          registry: ghcr.io
          username: ${{ github.actor }}
          password: ${{ secrets.GITHUB_TOKEN }}

      # 3. Executa o Deploy
      - name: Fazer Deploy com Docker Compose
        run: |
          docker-compose pull
          docker-compose up -d --no-build
          docker image prune -f
```

Configurando o Pipeline de CD

O objetivo desta fase é enviar nossos novos arquivos (docker-compose.yml, ci.yml atualizado, .gitignore atualizado) para o GitHub e "instalar" o novo pipeline na branch main.

1. **Faça o Commit das Mudanças:**

Primeiro, vamos criar a branch dev (se ela não existir) e enviar nossas mudanças para ela.

```
# (Se você já está na 'main', crie a 'dev' a partir dela)
git checkout -b dev
# Adicione todos os seus novos arquivos de configuração
git add .
git commit -m "Configura pipeline de CD (Job 2) e runner local"
# Envie a branch 'dev' para o GitHub git push origin dev
```

2. **"Instale" o Pipeline na main:**

Agora, precisamos que o ci.yml (com o Job 2) exista na main para que ele possa ser acionado por futuros merges.

No GitHub, abra um Pull Request de dev para main.

Observe (CI): O pipeline (Job 1) rodará e passará.

Faça o Merge deste PR.

Observe (CD): Assim que você fizer o merge, o pipeline completo (Job 1 e Job 2) será executado. Isso é o nosso primeiro deploy de instalação.

Verifique: Olhe seu terminal do runner local e acesse http://localhost:3000 para confirmar que o deploy de instalação funcionou.

Exercício em grupo

1. Conforme a prática desta aula, replique os passos em seu computador, na sequência altere a mensagem “Olá Mundo DevOps” nos arquivos app.js e app.test.js e faça um novo commit na Branch “dev”, o PR e finalmente a merge na main. Verifique se a mensagem foi alterada no servidor (sua maquina). Envie o link do repositório criado no github, onde será possível conferir a execução do pipeline CI/CD, juntamente com um pequeno parágrafo explicando em quais situações essa abordagem com CD pode ser útil dentro de uma empresa.

Bibliografia Básica

- PRESSMAN, Roger S. Engenharia de Software: uma abordagem profissional.8. ed. São Paulo: McCraw Hill – Artmedia, 2016.
- BOOCH, Grady; RUMBAUGH, James; JACOBSON, Ivar. UML
- Guia do Usuário. Rio de Janeiro: Campus, 2012.
- JOHNSON, Ralph; VLISSIDES, John; HELM, Richard; GAMMA, Erich. Padrões de Projeto. São Paulo: Bookman, 2008.