**Web Academy CI/CD**

**Laboratório - Aula 03**

**Objetivo do Laboratório**: Consolidar o uso de pipelines através da configuração de etapas de teste (usando o Jest) e com o uso de Containers.

**Pré-Requisitos:**

* Ter uma conta no GitHub
* Máquina com Linux / Windows instalado e Git já configurado com a conta do GitHub
* Ter uma conta no Docker Hub
* Arquivos disponíveis no Colabweb para o Laboratório (caso necessário)

**Exercício 01 - Aplicando o conceito de testes utilizando Jest com uma biblioteca externa para análise de cobertura**

1 - Crie um diretório “lab03\_01” para seu exercício.

2 - Instale o Typescript com o seguinte comando.

npm install typescript --save-dev

3 - Inicialize o projeto para criar o arquivo **tsconfig.json** (o Jest vai usar esse arquivo para compilar o código Typescript).

npx tsc --init

4 - Instale o Jest como dependência.

npm install --save-dev jest

5 - Inicialize a configuração do Jest com o comando

npx jest --init

Isso vai criar o arquivo jest.config.ts que vai ser usado pelo Jest para rodar os testes

Durante a inicialização, algumas perguntas vão ser feitas - responda sim a todas (configure o ambiente de teste como node e o provedor v8). Uma das perguntas é sobre fornecer os relatórios de cobertura de código.

6 - Adicione as seguintes bibliotecas que são utilizadas

npm install --save-dev ts-node ts-jest @types/jest

7 - Edite o seu arquivo jest.config.ts para o seguinte conteúdo

export default {

preset: 'ts-jest',

testEnvironment: 'node',

}

8 - Valide tudo executando o comando

npx jest

Se tudo der certo, ele vai avisar que nenhum teste foi encontrado.

9 - Crie o arquivo src/hello\_world.ts com o seguinte conteudo

const greeting = 'world';

export function hello(world: string = greeting): string {

return `Hello ${world}!`;

}

10 - Crei o arquivo src/hello\_world.test.ts com o seguinte conteúdo

import { hello } from "../src/hello\_world";

describe("test hello", () => {

it("should return hello world", () => {

expect(hello()).toBe("Hello world!");

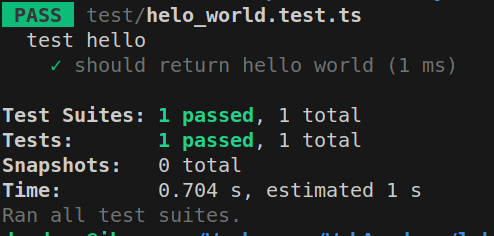
});

});

11 - Teste o projeto como comando

npx jest

O resultado vai ser algo do tipo…



12 - Atualize o conteúdo do arquivo hello\_world.ts

const greeting = 'world';

export function hello(world: string = greeting): string {

if (world == "WebAcademy") {

return "Hi WebAcademy!";

}

return `Hello ${world}!`;

}

13 - Ao rodar o teste novamente, teremos o mesmo resultado anterior, todavia, não estamos cobrindo todo o código! Para ver como está a cobertura de código, atualize o seu arquivo package.json, na parte de Debug, para o seguinte conteúdo

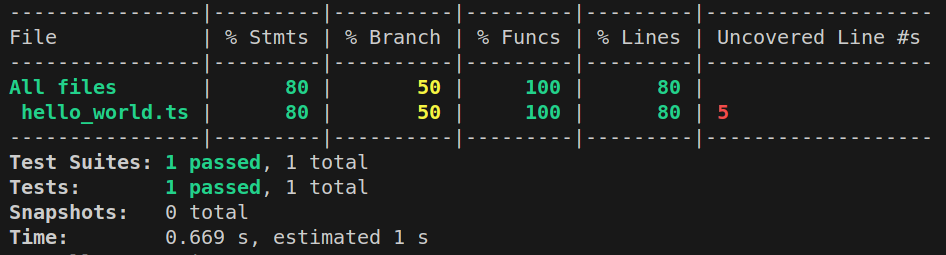
"scripts": {

"test": "jest --coverage"

}

}

Execute npm test e veja o relatório de cobertura



14 - Crie agora o diretório .github/workflows para armazenar o seu arquivo teste.yml e adicione o seguinte conteúdo

name: Test Pipe

on: push

jobs:

build:

runs-on: ubuntu-latest

strategy:

matrix:

node-version: [16.x, 18.x]

steps:

- name: Checkout repository

uses: actions/checkout@v3

- name: Set up Node.js ${{ matrix.node-version }}

uses: actions/setup-node@v3

with:

node-version: ${{ matrix.node-version }}

- name: Install dependencies

run: npm install

- name: Run the tests

run: npm test -- --coverage

15 - Crie um repositório público em sua conta no gitHub (como fizemos no lab anterior)

16 - Crie um arquivo .gitignore no seu projeto e configure para ignorar node\_modules

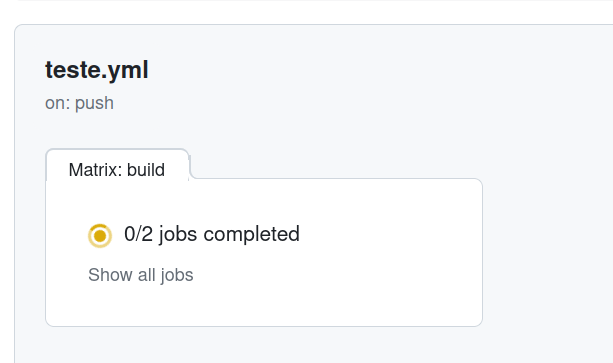
16 - Inicialize o seu projeto com o git, adicione todos os arquivos ao controle e faça o primeiro commit.

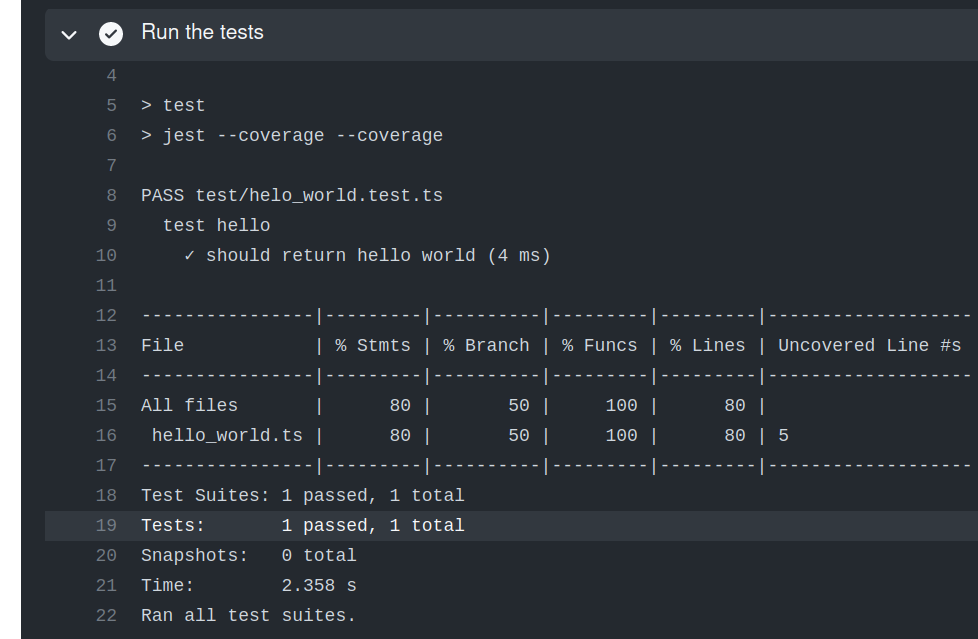
17 - Vá ao seu projeto local e associe ele ao seu repositório recém-criado

git remote add origin <link do seu repositório>

git push --set-upstream origin master

18 - Veja se o workflow ativou





PASSOS EXTRAS

19 - Vá em <https://codecov.io/> e configure uma conta em conjunto com a sua conta do GitHub e siga as instruções para configurar a chave de acesso nos segredos do seu repositório

20 - Adicione a seguinte etapa ao final do seu arquivo test.yml

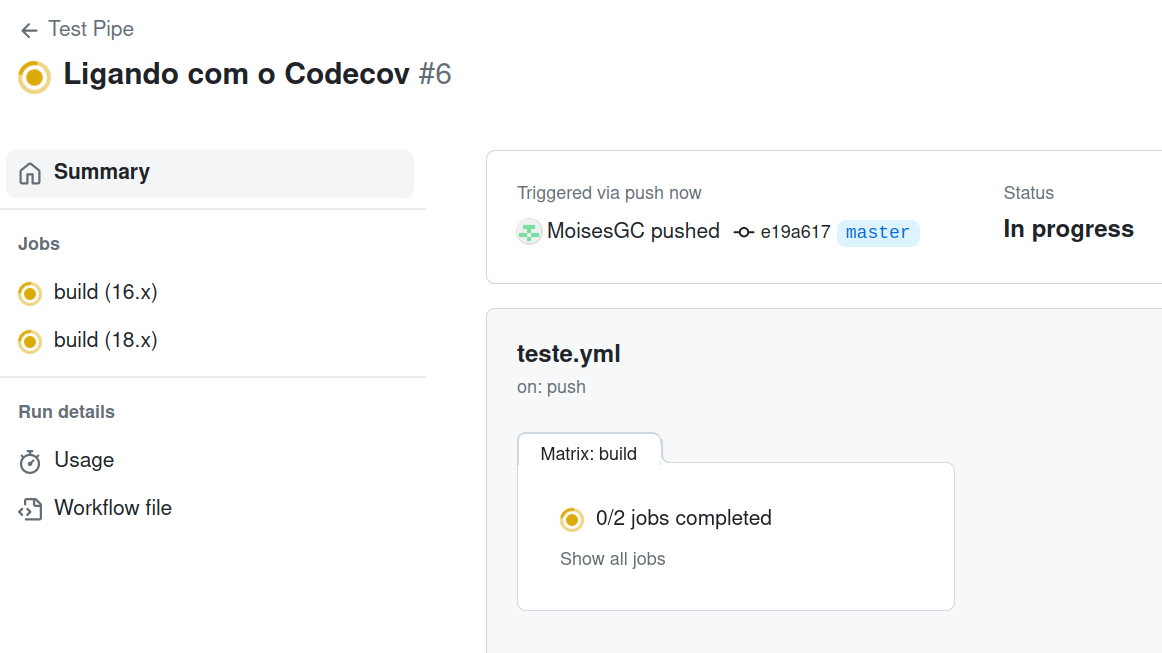
- name: Upload coverage reports to Codecov

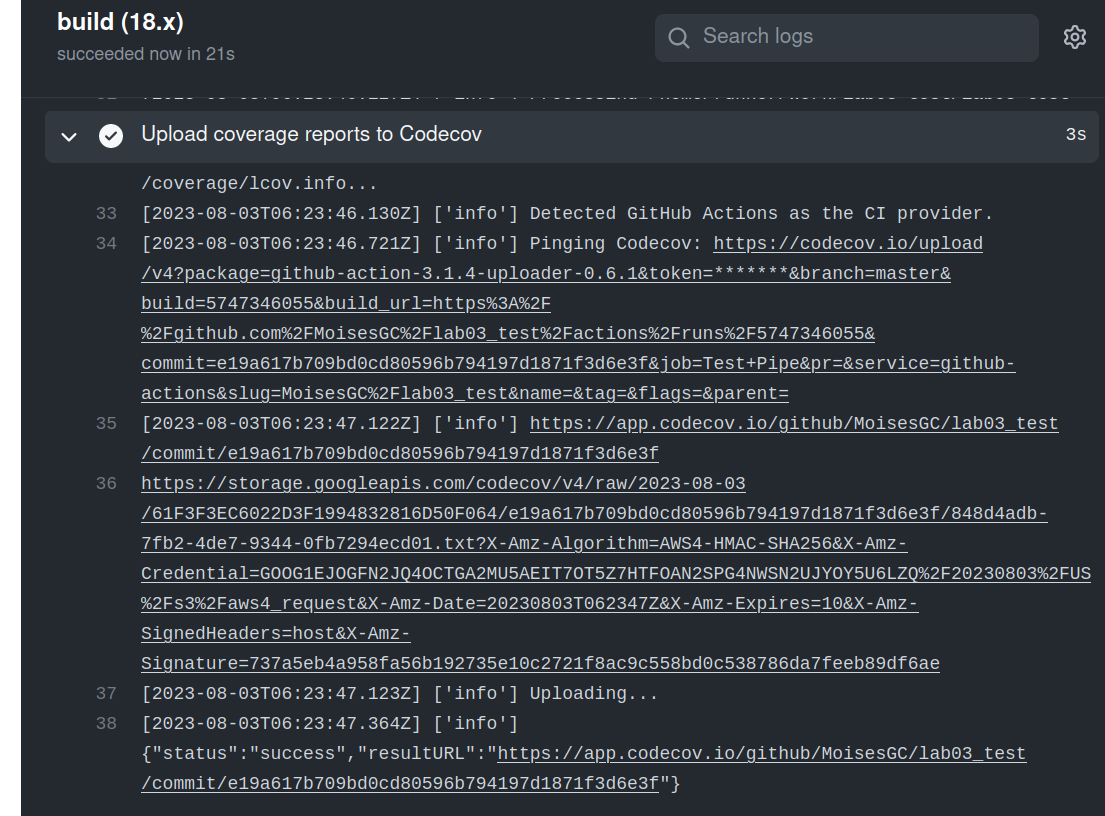
uses: codecov/codecov-action@v3

env:

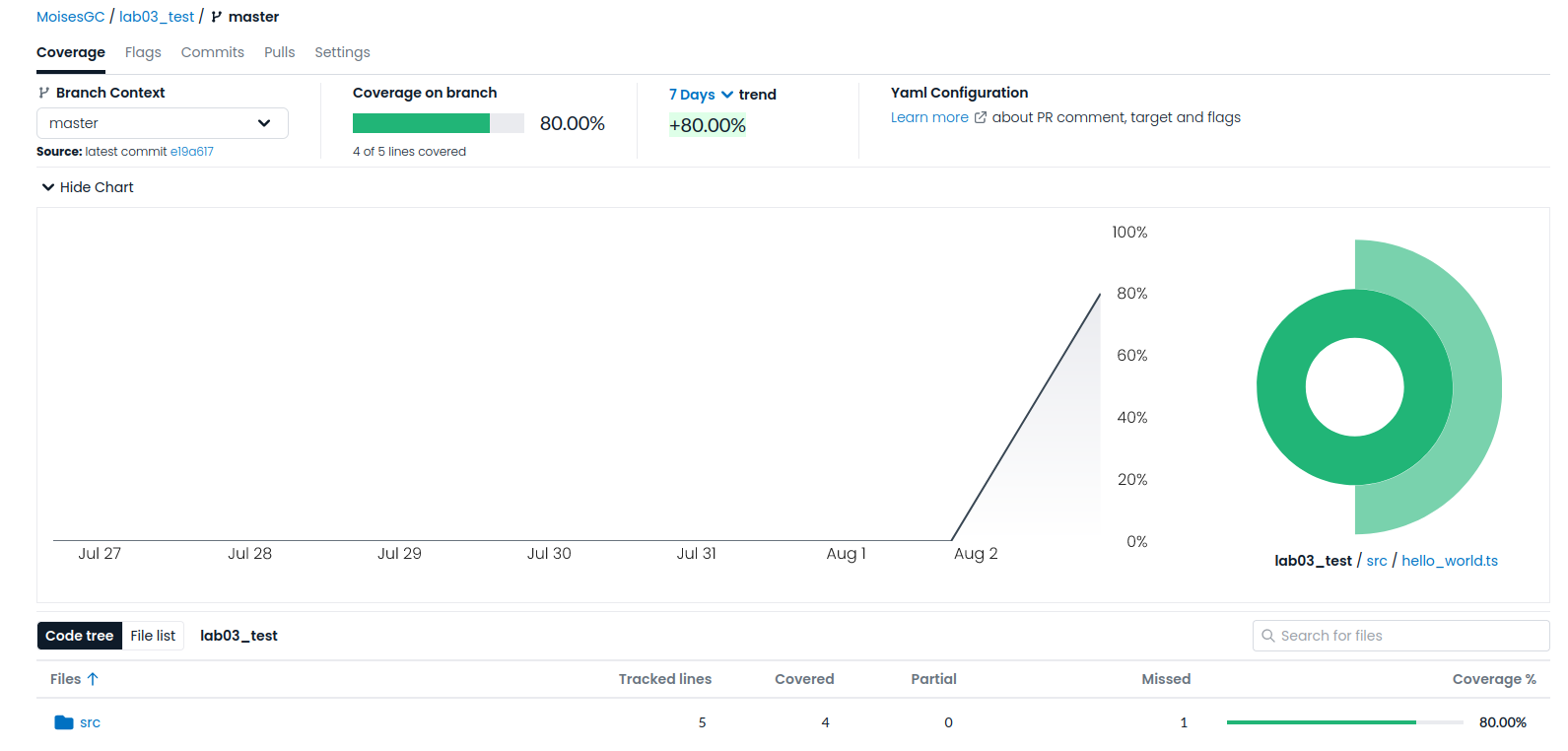
CODECOV\_TOKEN: ${{ secrets.CODECOV\_TOKEN }}

Faça o commit e depois atualize o seu repositório e observe a saída do passo de teste!





21 - Como tudo está ok, vá na sua conta e veja o relatório de cobertura do seu código!



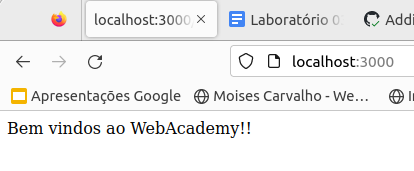
**Parabéns!! Você terminou o exercício!!**

**Exercício 02 - Usando containers com o GitHub Actions**

OBS: Caso não tenha o Docker configurado, siga as instruções para a sua versao de SO: https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-on-ubuntu-22-04

1 - Baixe o arquivo lab3\_docker.tgz do Colabweb e descompacte no seu diretório de trabalho.

2 - Rode *npm install* e depois *npm run start.* Abra um navegador na porta 3000 para confirmar que o projeto está rodando



3 - Com o seu Docker configurado, faça o build da aplicação

docker build . --tag web\_app

4 - Na sequência, rode a imagem criada no passo anterior - se tudo estiver certo, deve funcionar do mesmo jeito que a aplicação local - use o comando abaixo para rodar a imagem docker

docker run -it -p 3000:3000 web\_app

5 - Crie um arquivo .gitignore no seu projeto e configure para ignorar node\_modules

6 - Crie agora o diretório .github/workflows para armazenar o seu arquivo docker\_build..yml e adicione o seguinte conteúdo

name: Build on DockerHub

on:

push:

env:

DOCKERHUB\_USERNAME: ${{ secrets.DOCKERHUB\_USERNAME }}

DOCKERHUB\_TOKEN: ${{ secrets.DOCKERHUB\_TOKEN }}

jobs:

build:

runs-on: ubuntu-latest

steps:

- name: Checkout code

uses: actions/checkout@v3

- name: Set up Docker Buildx

uses: docker/setup-buildx-action@v3

- name: Login to DockerHub

uses: docker/login-action@v3

with:

username: ${{ secrets.DOCKERHUB\_USERNAME }}

password: ${{ secrets.DOCKERHUB\_TOKEN }}

- name: Build and push Docker image

uses: docker/build-push-action@v5

with:

context: ./

push: true

#dockerfile: ./Dockerfile # Specify the path to your Dockerfile

tags: moisesgc/webacademy:latest

deploy:

runs-on: ubuntu-latest

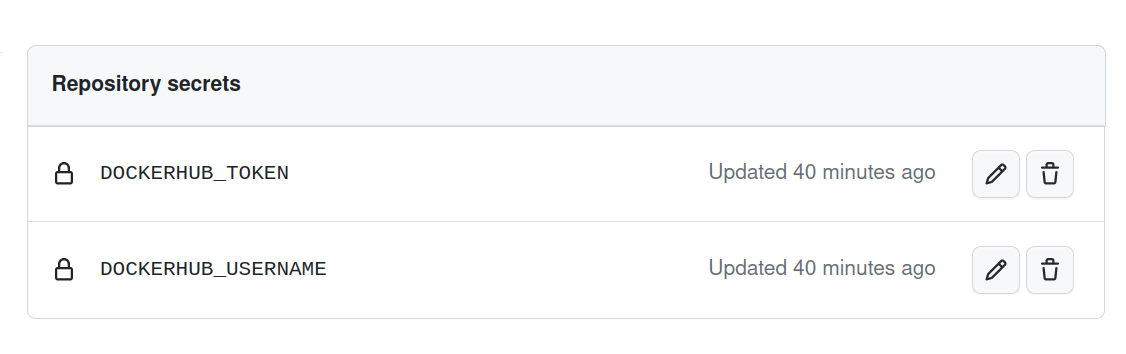
steps:

- name: Fazendo deploy

run: echo "Deploy com sucesso!"

7 - Crie um repositório público em sua conta no gitHub (como fizemos no lab anterior)

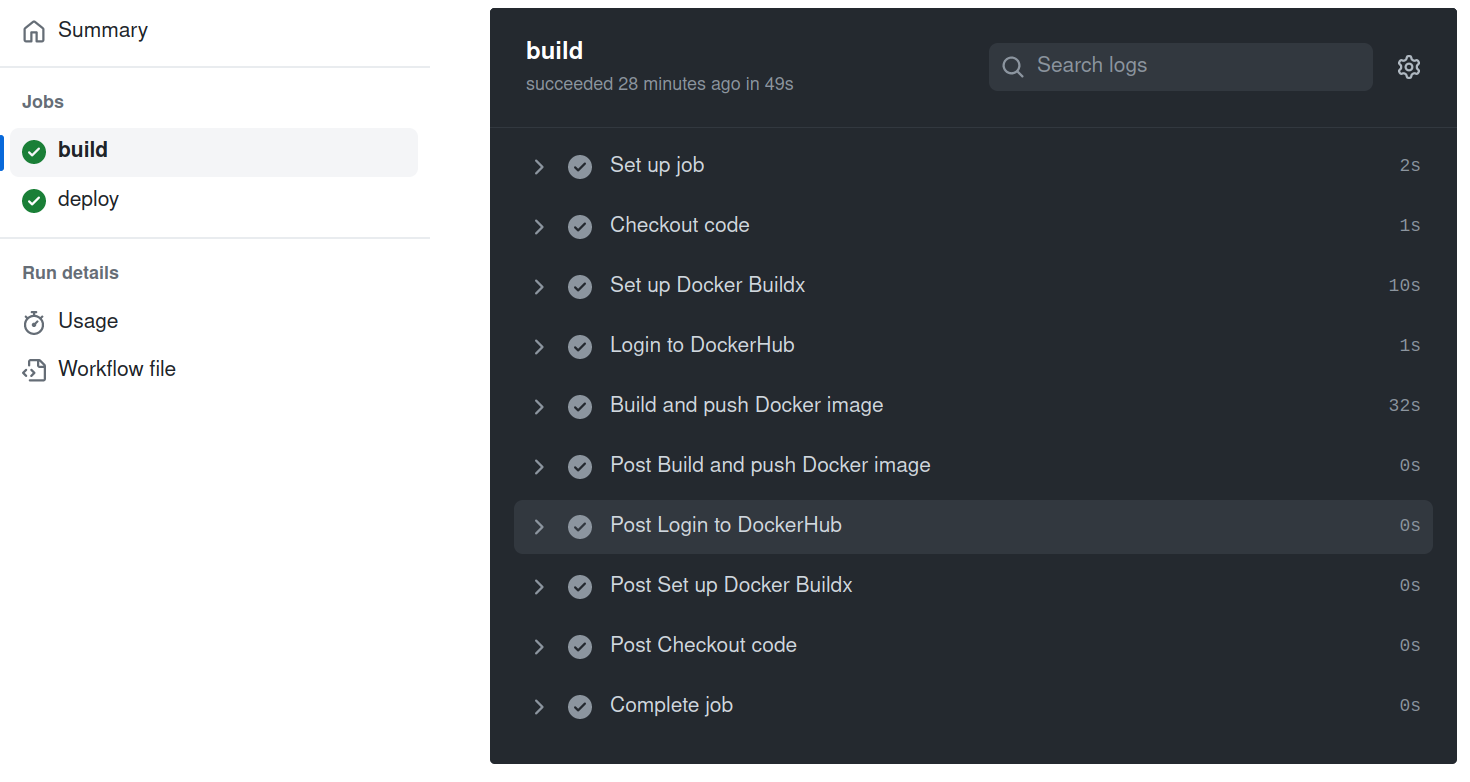
8 - Vá na seção de configuração do seu repositório e adicione as suas credenciais do DockerHub (se não tiver, vá em <https://hub.docker.com/> e crie a sua conta e gere a chave)

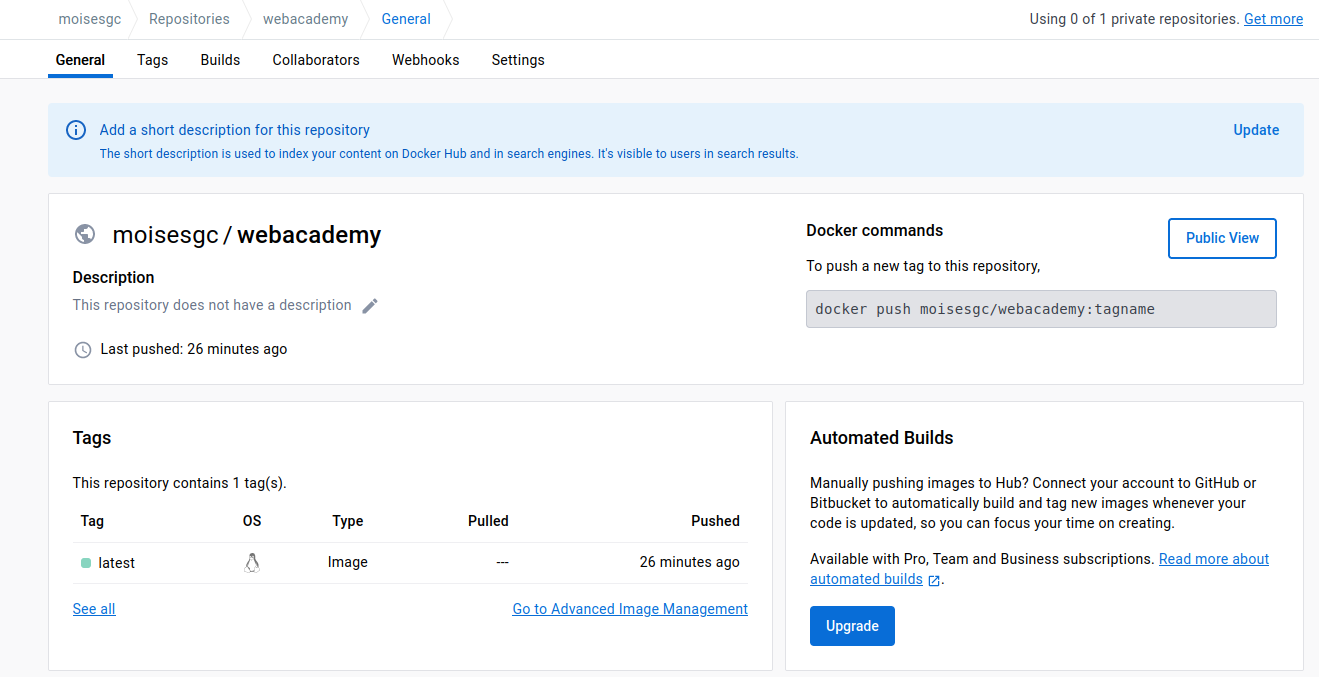


\* com a sua conta do DockerHub criada, atualize o arquivo docker-compose.yml com o nome do seu usuário do DockerHub.

9 - Inicialize o seu projeto com o git, adicione todos os arquivos ao controle e faça o primeiro commit.

10 - Se tudo funcionar, não haveŕa erros em seu build e a sua imagem estará disponível na sua conta do Docker Hub!





**Parabéns, você terminou o exercício!!**