Miniprojeto M1A

#TDW #MCTW

URL Gitlab: https://gitlab.com/pw-mctw/tdw-2022/tdw-mp1-miguel-teixeira

URL GitHub Actions: https://github.com/Micatalyst/tdw-mp1-miguel-teixeira

Credenciais GitHub Actions:

Mail – Miguel.melo.teixeira@ua.pt

Pass – miguel_mp1a

Miguel Teixeira nº 100402

| Apresentação do desafio e da plataforma de CI / CD escolhida

Para o desenvolvimento deste miniprojeto foi-me lançado o desafio de desenvolver uma pipeline de integração contínua, para um projeto já existente, no gitlab e numa outra plataforma de CI / CD à minha escolha.

A plataforma que tive a liberdade de escolher foi o GitHub Actions.

| Implementação técnica

Inicialmente, decidi transferir o template disponibilizado para o projeto e quando tentei correr o projeto, deparei-me com algum problema relativo ao comando "npm run dev" e por isso, decidi criar um projeto de next.js à parte e passar para o miniprojeto apenas os ficheiros relevantes ao next.js que podiam não vir totalmente instalados com os restantes ficheiros do template, tais como os node_modules.

Logo de seguida, também reparei que havia um erro com os módulos do tailwindcss e decidi correr um comando para instalar o tailwindcss do 0 de forma a colmatar todos os erros possíveis por falta de módulos.

Depois de ter certeza de que o projeto estava estável e funcional a nível local, decidi iniciar um repositório de Git na mesma pasta do projeto e chamei a este repositório de "repo_mp1a". Fiz a ligação com o repositório do gitlab remoto e dei um push inicial dos conteúdos do template do projeto. Também fui ao ficheiro o .gitignore, adicionei o ficheiro .idea e retirei o ficheiro out. Depois disto, criei as branches que ia usar e também dei push para o gitlab.

Depois, corri o comando "npm run lint" e deparei-me com alterações que também tive de fazer ao código relativamente aos <link> do react router. Tive de os adicionar e retirar os "href" dos <a> e passar para os <link> de forma a evitar um reload total, desnecessário, na página sempre que estes links tivessem de ser acessados.

De seguida, nesta fase, instalei o prettier e o jest onde os guardei na parte dos scrips presentes no package.json. Por fim, aqui, instalei o lefthook e configurei o seu ficheiro .yml para correr uma verificação no pre-commit e no pre-push.

Seguidamente, corri um "npm run build" e obtive uma primeira build do projeto na pasta out. Arrastei esta para o netlify e configurei o meu custom domain, guardando posteriormente algumas credenciais, tais como, o CNAME, SITE ID e ACCESS TOKEN e de seguida, configurei o DNS na Cloudflare de forma a conseguir adicionar um domínio personalizado ao netlify.

Depois do netlify, comecei a configurar o contentful, instalando o Gatsby CLI tool. Com isto, perdi algum tempo a tentar resolver novos erros que me apareceram como por exemplo, o mesmo problema dos <link> e dos <a> inicial e, mais tarde, um problema relativo à width das imagens que deveriam estar a ser carregadas pelo contentful no projeto. No final consegui resolver alterando a versão que eu tinha do next.js para uma versão mais estável e antiga, a 12.3.1.

De seguida, guardei as variáveis relativas às credenciais do contentful, na secção privada das variáveis do gitlab, para que desse modo, quando desse trigger na pipeline e deploy para o netlify, o conteúdo do contentful aparecesse na página do projeto.

Com isto feito, comecei a desenvolver o primeiro trigger personalizado da pipeline com webhooks no contentful, para que sempre que o conteúdo do contentful atualizasse, também pudesse dar trigger da pipeline no gitlab e gerar uma nova build com o novo conteúdo para o netlify.

Quanto ao segundo trigger, tive de recorrer aos schedules do gitlab, com o objetivo de dar um trigger temporal. Sempre que a hora 00:00 chegasse a minha pipeline podia dar trigger.

Finalizando esta parte e passando para uma nova plataforma de CI / CD, clonei o meu repositório para o GitHub. Associei a minha chave pública SSH ao GitHub e criei um repositório novo local ao qual chamei de "repo_mp1a_actions". Tentei fazer um commit e tudo funcionou perfeitamente.

A partir daqui tudo foi um pouco mais complicado porque tive que ler alguma documentação e aprender a utilizar o workflow do GitHub Actions. Com a documentação e ajuda de alguns vídeos, consegui começar a perceber a syntax das actions do GitHub podendo começar a achar alguns padrões em algumas partes e entendendo o que seriam os jobs e como poderia correr scrips neles. Este processo de assimilação de novos conteúdos foi algo um pouco mais demorado e com vários erros pelo caminho, mas consegui perceber as bases e, um pouco melhor, de que forma poderia conseguir implementar a minha pipeline.

A certa altura, depois de ter uma boa base para a minha action, comecei a fazer um webhook para o GitHub para chamar posts do contentful. Seguindo toda a documentação que achei, gerei um token específico dentro do GitHub ao qual precisei de, de uma forma diferenciada, inserir no contentful.

Nesta fase tive bastantes problemas porque não conseguia entender verdadeiramente qual era o problema do meu código não estar a ser apresentado como devia pelo GitHub Actions. Uma solução que achei foi inserir nomes para os artefactos nas minhas actions e subir a versão do node a ser utilizado, mas isto não resolveu tudo. O problema, de não aparecerem posts na minha página do netlify, continuava, então decidi reescrever as variáveis ou "secrets" que tinha inserido no GitHub para a sua conexão com o contentful. Isto surpreendentemente resultou e desta forma finalizei o projeto.

| Diferenças observadas entre plataformas

Para começar o ficheiro relativo à pipeline, no gitlab fica na raiz do projeto e no GitHub Actions fica dentro de 2 pastas (.github/workflows).

Depois posso observar que, no gitlab, os jobs apenas são chamados de imediato por um nome no canto à esquerda da indentação. No GitHub Actions, precisei de ter em primeiro lugar "jobs" a ser chamado e depois dentro, a lista de jobs propriamente dita.

Para conseguir organizar os jobs de forma sequenciais no gitlab, tive de apenas na raiz escrever "stages" e ordenar dentro em lista. No GitHub Actions, tive de usar algo chamado de "needs" ao qual inseri a última dependência na action. Ou seja, se eu queria que o build começasse depois dos testes, tive de inserir nos seus "needs" os testes propriamente ditos.

Mais observações que posso fazer é o facto de que, no gitlab para definir uma imagem chamamos de "image" e especificamos o node e a sua versão. No GitHub Actions, começamos por inserir "runs-on" e especificamos o sistema operativo e depois dentro em "uses" chamamos a action "setup-node" e ainda adicionamos um "with" onde dentro inserimos um "node-version" e especificamos aqui a versão do node.

Já para correr scrips / comandos no gitlab usamos "script" e no GitHub Actions usamos "run"

Para as variáveis no gitlab podemos nas definições relativas a CI / CD, adicionar estar variáveis secretas. No GitHub Actions, vamos a secrets e até podemos personalizar por environment (ambiente).

Tirando também a interface de cada plataforma, outras funções são similares como definir caminhos "paths" (gitlab) vs "path" (GitHub).

| Principais desafios e obstáculos não ultrapassados

Como principais desafios tive as conexões com o contentful porque senti que nas duas vezes que tive mais problemas relacionados ao contentful no final das contas e de perder bastante tempo, tive problemas de compatibilidade de versões e variáveis de credenciais que não estavam a ser chamadas corretamente. Quando a obstáculos não ultrapassados, penso que possa ter sido principalmente a melhor otimização da pipeline em ambas as plataformas.

| Conclusões

Este projeto de CI / CD ajudou-me a perceber que realmente existe muito para além do que é apenas construir uma página e também ajudou-me a perceber a realidade sobre a complexidade que é gerir projetos enormes num contexto que poderia estar presente no mundo do trabalho.