E finalmente o front-end

Terminando! Agora vamos fazer a parte do *front-end*, temos apenas 1 para subirmos: React.

No Gitlab, crie um repositório novo, pode ser um projeto em branco mesmo, se possível com o nome "react" caso você já não tenha utilizado este nome de projeto no seu repositório (caso contrário escolha outro nome de projeto). Importante você criar isso no **SEU** repositório hein!

Você pode opcionalmente descrever o que será seu repositório, no campo "Project deployment target (optional) não precisa alterar nada e no campo "Visibility Level" pode manter a opção "Private" mesmo (à não ser que você queira deixar seu repositório público, por sua conta e risco). Também pode manter checada a opção "Initialize repository with a README" e clique em "Create project".

Faça o git clone para a sua máquina, pois traremos algumas coisas já pré-prontas!

Eu sei que subimos lá atrás o nosso reactsite, neste caso vamos fazer igual ao que fizemos com o back-end, então pode manter o container e a imagem sem problema nenhum!

Aproveite para já fazer algumas configurações de praxe:

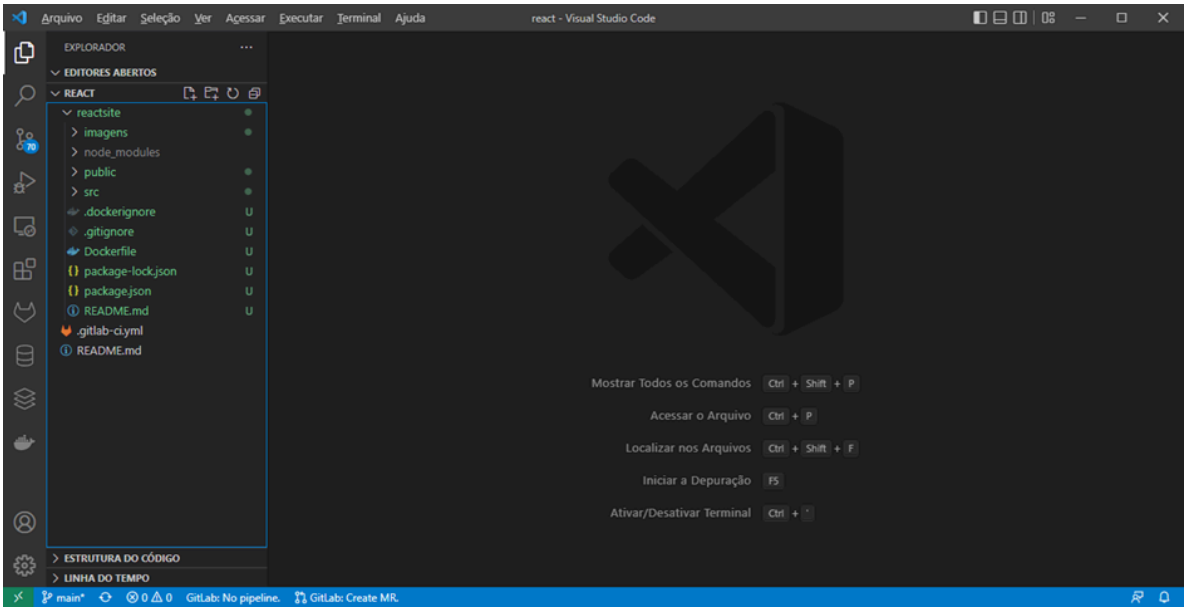
No menu lateral à esquerda vá em "Settings" e clique em "CI/CD", depois clique no botão "Expand" da guia "Runners". Habilite o nosso "runner" para este projeto e desabilite o "shared runner"

Vá ao repositório https://gitlab.com/everton.juniti/descomplica, e copie a pasta cicd_frontend\reactsite para o seu repositório do React. Traga desse mesmo

repositório na pasta cicd_frontend\01-Build_React o seguinte arquivo:

.gitlab-ci.yml: pode deixar na raiz do seu repositório do React

Deverá ficar algo assim:



Pode fazer o push para o seu repositório do React!

Observe que assim que você fez o push, nenhuma pipeline chegou a rodar.

Isso acontece pois neste exemplo do React não há nenhum teste unitário, então não fazia sentido incluir este job.

Observação: não é uma boa prática não ter testes unitários em aplicações, como aqui é mais didático estamos utilizando um exemplo proposital sem testes unitários.

Mas segue a mesma regra de Continuous Delivery e Continuous Deployment do Java e do Python, ou seja, a pipeline só é iniciada quando você criar uma branch release ou fizer um merge request para uma branch com este nome. Está na instrução only em todos os jobs.

Veja que o script de pipeline do React é bastante semelhante à do Python:

```
variables:
    APP_PATH: reactsite
    DOCKER_TAG_NAME: $APP_PATH:latest
    NOME_DO_CONTAINER: MeuReactSite
    NOME_DA_REDE: MinhaRede
    BIND_DE_PORTA: 3000:3000

stages:
    - prebuild
    - build
    - deploy

Limpar_Imagem:
    stage: prebuild
    only:
      - /^release_[0-9]+(?:.[0-9]+)+$/
    variables:
        GIT_STRATEGY: none
    script:
        - docker container stop $(docker container ls -a -f
ancestor=$DOCKER_TAG_NAME -q) || FAILED=true
        - echo Tentativa de parada de container. Falhou? - $FAILED
        - FAILED=false
        - docker container rm $(docker container ls -a -f
ancestor=$DOCKER_TAG_NAME -q) || FAILED=true
        - echo Tentativa de remoção de container. Falhou? - $FAILED
        - FAILED=false
        - docker image rm $DOCKER_TAG_NAME || FAILED=true
        - echo Tentativa de remoção da image. Falhou? - $FAILED

Criar_Imagem:
    stage: build
    only:
      - /^release_[0-9]+(?:.[0-9]+)+$/
    script:
        - docker build --no-cache -t $DOCKER_TAG_NAME ./$APP_PATH

Criar_Container:
    stage: deploy
    only:
      - /^release_[0-9]+(?:.[0-9]+)+$/
    variables:
        GIT_STRATEGY: none
    script:
        - docker run --name $NOME_DO_CONTAINER --network $NOME_DA_REDE -p
$BIND_DE_PORTA -d $DOCKER_TAG_NAME
```

A única diferença aqui é o nome do container, o bind de porta e a tag de imagem, o restante é basicamente igual!

Essa é uma característica interessante de pipelines, que é o reuso. Praticamente alteramos apenas algumas variáveis mas o restante do script é basicamente o mesmo.

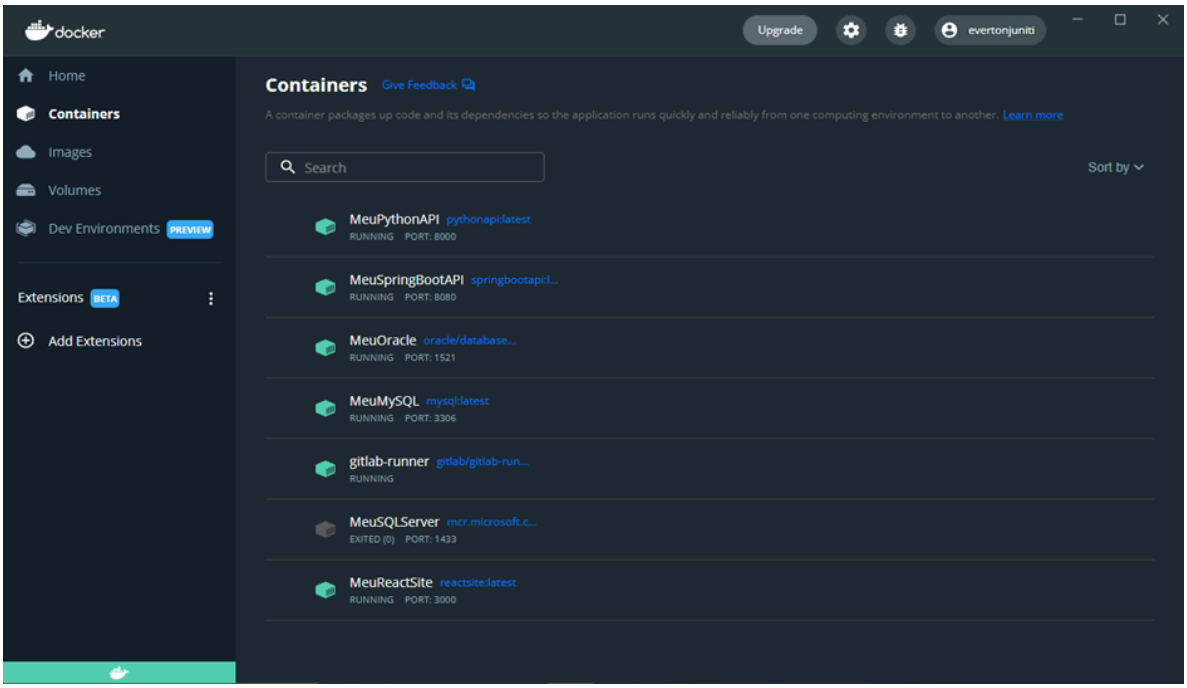
Agora é a hora da verdade! Vamos subir nosso último pipeline! Mas antes:

Verifique se os bancos de dados MeuOracle e MeuMySQL estão “de pé”.

Verifique se as nossas APIs em Java e Python estão “de pé”.

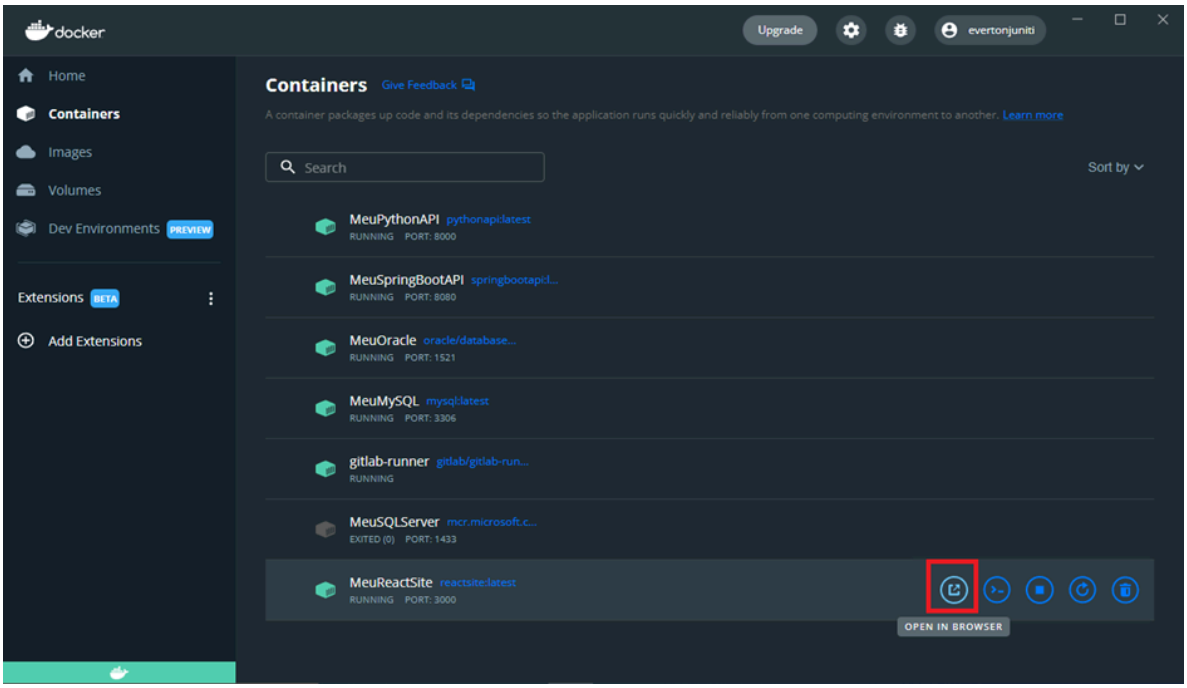
No repositório do React, crie uma branch nova à partir da main, chamada release_1.0. Assim que você criar a branch a pipeline já se iniciará.

O resultado final são todos os nossos containers de pé:



Experimente alterar alguma coisa do repositório na branch main e crie uma branch nova (ex: release_1.1) ou faça um merge request da branch main para a branch release_1.0 que criamos! Vamos ver que também conseguimos trocar a imagem e o container assim como fizemos com o back-end!

Agora é hora da festa! Vamos testar toda a solução! Abra o site do React, você pode fazer de 2 formas: abrindo o browser e digitando localhost:[porta que você fez o bind] (no meu caso é localhost:3000) ou indo no painel de containers do Docker Desktop e apontando o mouse em cima do container do React, vai aparecer alguns ícones, o primeiro da esquerda para a direita é o "OPEN IN BROWSER", que abre o browser já com o endereço certo para acesso.



Navegue no site para efetuar cadastros ou alterar qualquer um que você já tenha feito!

Note que o site irá chamar as APIs tanto do Java quanto do Python.

Estes por sua vez irão interagir com os bancos de dados (respectivamente o Oracle e o MySQL).

Você poderá abrir o Oracle SQL Developer e o MySQL Workbench e fazer consultas nas tabelas para verificar se as mesmas estão sendo preenchidas pelas chamadas das APIs que por sua vez são chamadas pelo site.

Temos uma solução de ponta-a-ponta que subirmos via containers do Docker, estes por sua vez foram criadas através de pipelines do Gitlab.

Vimos na prática a facilidade de subir tanto a camada de dados (os bancos de dados) quanto a camada de domínio (as aplicações em Java e Python) quanto a camada de apresentação (o nosso site) e como podemos utilizar as automações que criamos através das pipelines para diferentes propósitos:

Importante lembrar que o "runner" ficará "escutando" a todos os repositórios ao qual foi registrado, então desligar o container do "runner" significará que não haverá "ninguém" para executar quaisquer dessas pipelines.

Aqui no nosso exemplo criamos certa dependência entre as soluções: o site depende das APIs e as APIs dependem de seus bancos de dados, qualquer que seja o container que você desligue, acabará desligando alguma parte da solução.

- No caso dos bancos de dados, o nosso propósito era subir e manter os containers de pé. Caso algum container seja desligado, podemos rodar a pipeline que tentará ligar os containers

- No caso da API em Java, podemos exercitar a integração contínua através do feedback dos testes unitários, para corrigir rapidamente eventuais erros

- No Java, Python e React vimos como fazer a entrega contínua e implantação contínua no nosso Docker local partindo das pipelines executadas pelo nosso "runner" local também

Atividade Extra

Para se aprofundar no assunto desta aula releia o capítulo: "4 *Pipeline* de implantação", do livro "Jornada Devops 2ª edição", de Antonio Muniz, Analia Irigoyen, Rodrigo Santos e Rodrigo Moutinho.

Pense no conceito trazido pelo capítulo do livro e faça um paralelo com as implantações que fizemos com as pipelines de ponta-a-ponta.

Adicionalmente veja o documento de referência: "GitLab CI/CD include examples" que mostra como apartar um script de pipeline em vários arquivos.

Link do documento: https://docs.gitlab.com/ee/ci/yaml/includes.html

Referência Bibliográfica

OGURA, Everton J. **Repositório do GitLab, projeto Descomplica.** Disponível em https://gitlab.com/everton.juniti/descomplica. Acesso em 18 de junho de 2022.

Ir para exercício