Gitlab CI/CD: Introdução

O Gitlab CI/CD é uma ferramenta para desenvolvimento de software que usa as metodologias de integração contínua (Continuous Integration), entrega contínua (Continuous Delivery) e implantação contínua (Continuous Deployment).

Essas metodologias são alcançadas com algumas práticas através da ferramenta Gitlab, ou seja, não é só por ter um repositório no Gitlab que você já está aplicando essas 3 metodologias na prática!

Olhando somente o aspecto de integração contínua, se você usa o Gitlab como repositório para subir código várias vezes por semana, várias vezes ao dia até e se consegue compilar e testar sua aplicação com scripts (ao invés de dar "play" no seu computador) então você estará utilizando esta primeira parte da metodologia.

Os scripts automatizados para compilar e testar o código fazem com que você tenha um feedback rápido do que está construindo e corrija rápido, assim você diminui as chances de enviar algo com erros para a produção.

Já a entrega contínua é a prática de integração contínua somada à implantação automatizada através de scripts, assim você poderá enviar o teu código para ambientes de desenvolvimento, homologação e produção de forma contínua, várias vezes na semana ou várias vezes no mesmo dia. Você automatiza esse processo através de scripts, mas ainda há a necessidade de intervenção humana para disparar esses scripts.

O último é a implantação contínua, que é a prática de entrega contínua só que totalmente automatizada, ou seja, o disparo do script de entrega nos ambientes é

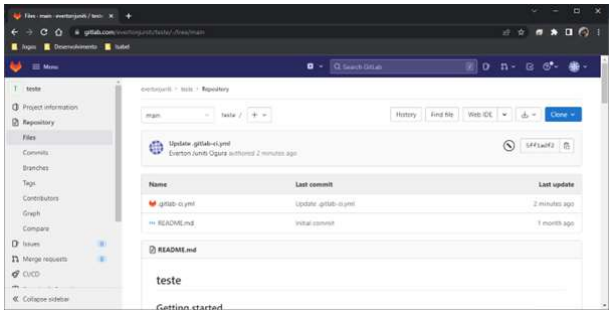
feito automaticamente sem intervenção humana, através de regras que você inclui nos scripts de automação.

Veremos conceitualmente como o Gitlab nos permite rodar esses scripts! Um script é literalmente um arquivo com um passo-a-passo, como uma receita de bolo que diz o que tem que ser feito. Nós já vimos o quão poderosos são o scripts como um arquivo Dockerfile, que nos diz passo-a-passo como fazer a construção de uma imagem Docker, aqui um script do Gitlab se chama ".gitlab-ci.yml" e deve sempre ter essa nomenclatura, começando com um "." (ponto) mesmo e escrito todo em minúsculas. Só pode haver um único arquivo desse por repositório do Gitlab, portanto o script refere-se apenas ao repositório em que ele exista, um repositório que possui este script já estará habilitado para executar automações.

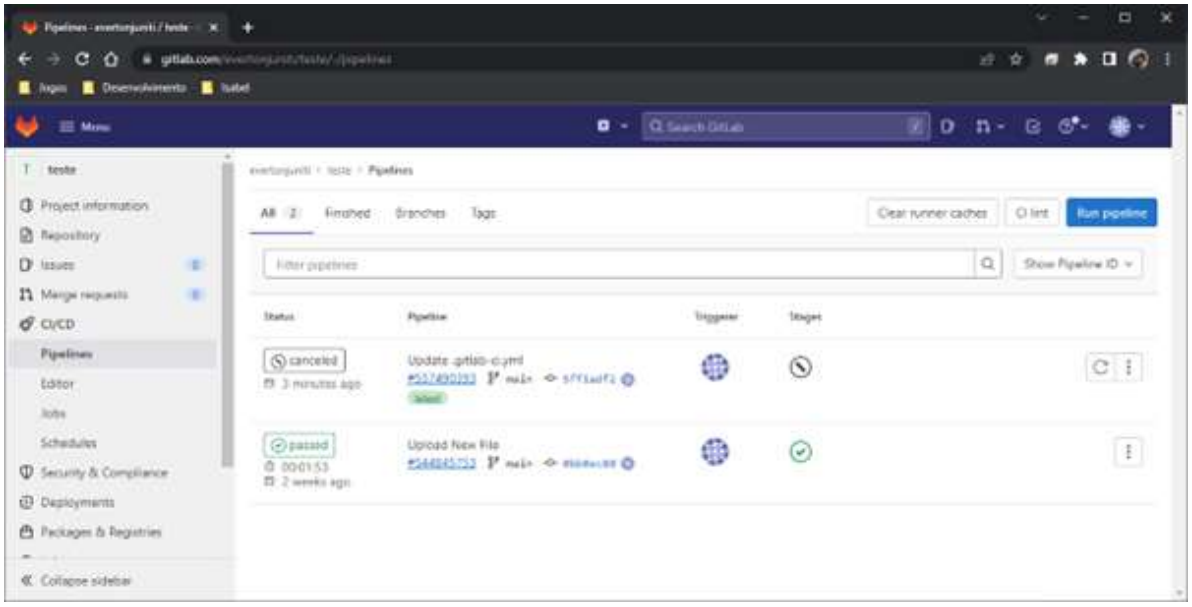
Então vamos lá: **você deverá criar um repositório novo no Gitlab** para os testes que faremos nesta aula, pode criar com qualquer nome (não fará diferença), mas é interessante criar um repositório apartado de qualquer outro que você já tenha para não "bagunçarmos a casa", já que serão feitos vários testes de "pipeline" aqui e em outras aulas!

Neste momento não veremos como criar um script ".gitlab-ci.yml", mas veremos conceitualmente como ele nos ajuda a criar uma automação no Gitlab para trabalhar com o código-fonte que estiver presente no repositório.

A presença de um script com uma receita do que deve ser feito fará com que o Gitlab crie automaticamente o que chamamos de "pipeline", exemplificando isso eu criei um repositório de teste e incluí um script com algumas instruções (que neste momento não são importantes). Um "pipeline" neste sentido pode ser entendido como um fluxo que trafega um entregável partindo de origens como código-fonte passando por várias etapas até ser entregue como produto final em algum lugar.



Como este meu script possui uma receita (as instruções do que deve ser feito), automaticamente foi criado um “pipeline” de automação, que pode ser visto no menu à esquerda do Gitlab em “CI/CD” e no sub-menu “Pipelines”

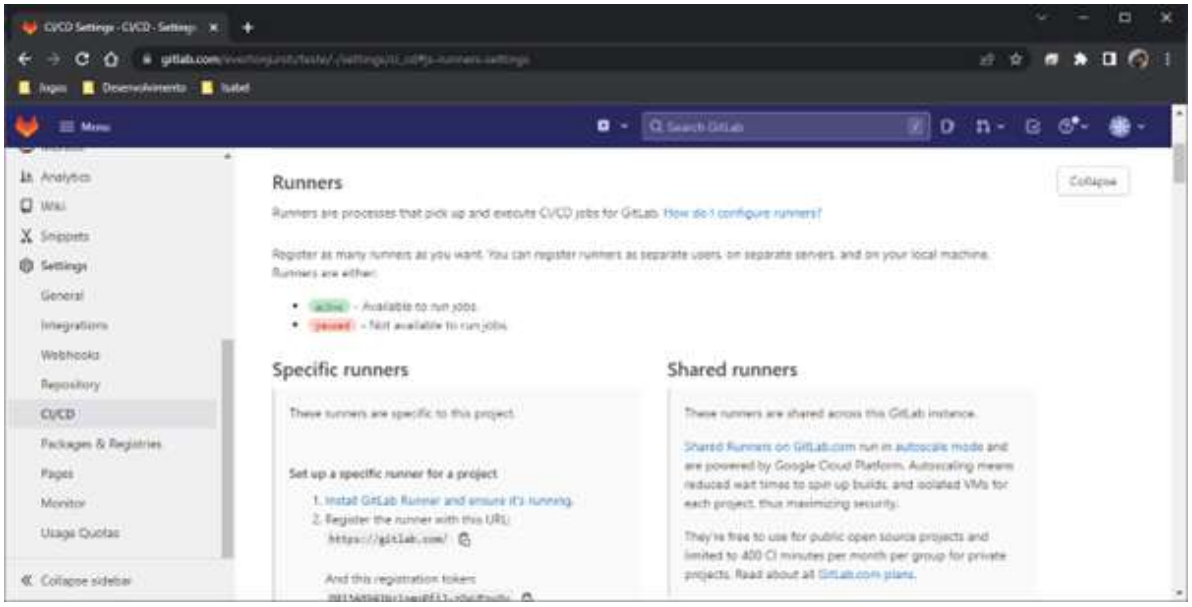


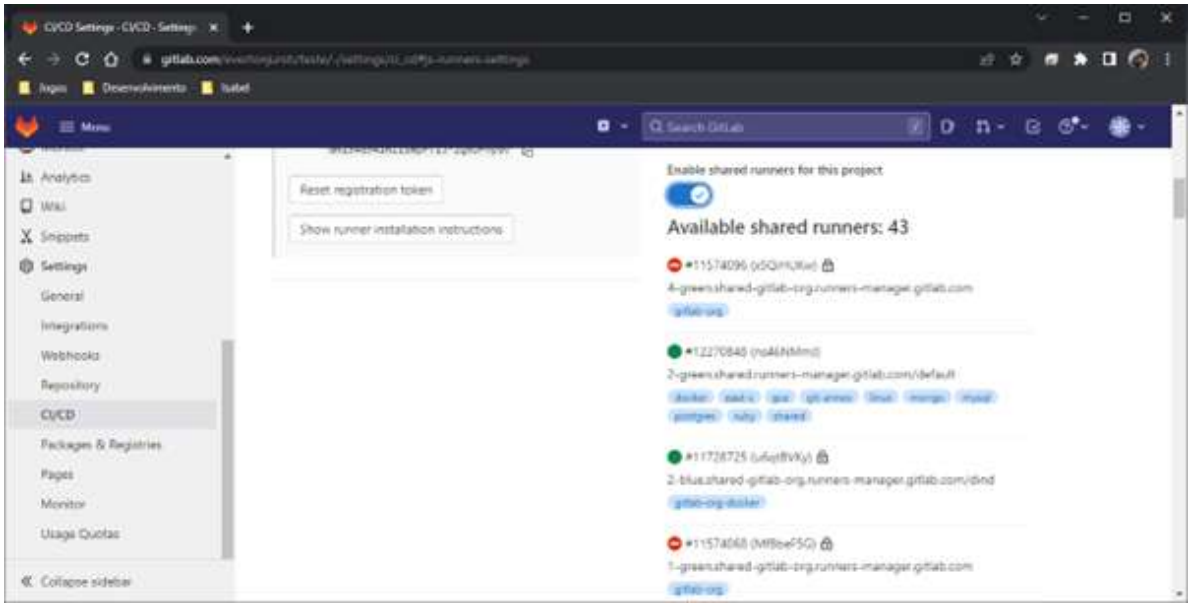
Com um arquivo de script válido (com instruções) no repositório, toda e qualquer alteração no repositório criará um pipeline, que fará com que este script de automação seja executado. Mas quem é que executa esses scripts afinal?

Apresentando: Gitlab Runners!

Um robô com uma estrutura mínima é quem executa esses scripts, ele é preparado para ler o script e executar o que está dentro do script. Você pode checar em seu repositório quais “runners” estão habilitados para executar suas “pipelines”, vá no menu à esquerda no item “Settings” e depois no sub-menu “CI/CD”, por último na

página que for exibida tem uma guia chamada “Runners”, aperte o botão “Expand” ao lado do nome “Runners” para exibir todos os robôs que atendem seu repositório.





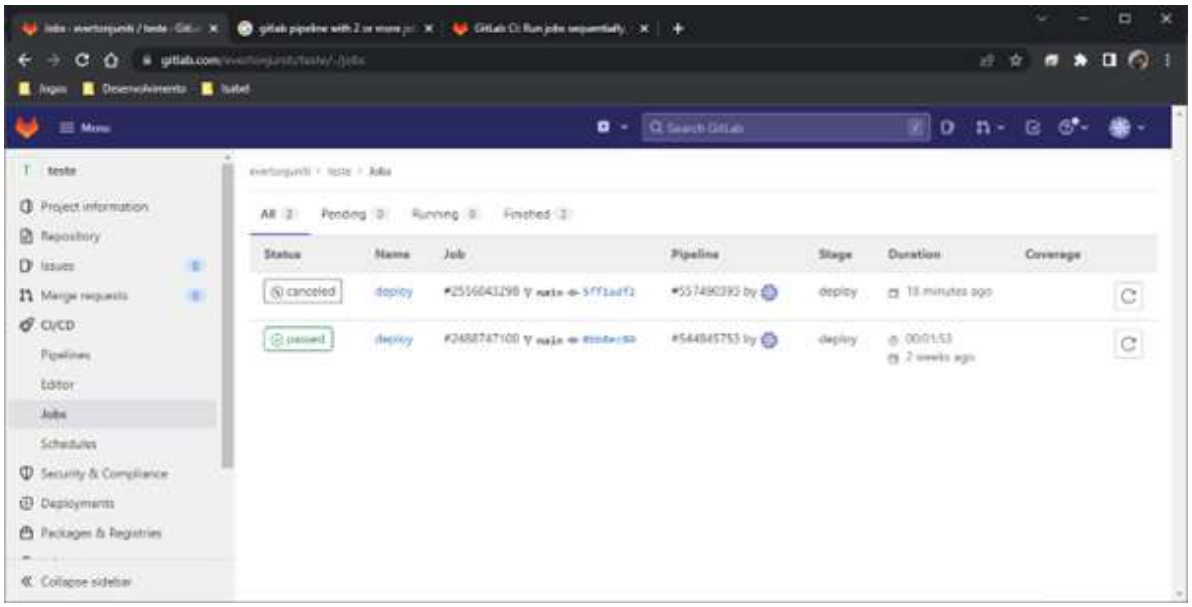
No exemplo acima, há vários “runners” habilitados, pelo menos os que estão com uma bolinha verde à frente podem ser utilizados, os que estão com uma bolinha vermelha estão offline (portanto inutilizáveis) e os que tem um triângulo cinza escuro estão obsoletos (portanto inutilizáveis também). Outro ponto interessante é que neste caso estão habilitados o que chamamos de “shared runners”, ou seja, são robôs compartilhados com vários repositórios, são robôs do próprio Gitlab que executam as pipelines desses vários repositórios.

Esses robôs podem ter várias capacidades como copiar arquivos do repositório, rodar comandos para compilar projetos, rodar comandos para construir imagens do Docker e subir containers e etc, mas tudo depende de como os robôs forem disponibilizados e neste modelo de robôs compartilhados pode ser que esses robôs não satisfaçam nossas necessidades.

Outra coisa importante na execução de pipelines é que você pode ter o que chamamos de "Jobs" dentro do script de uma pipeline. "Jobs" são pequenas tarefas apartadas dentro de um script, que podem ser executadas sequencialmente ou paralelamente. Significa que você pode ter um script com pequenos pedaços cada um fazendo alguma coisa como: "job" de teste, "job" de compilação, "job" de implantação, não precisa tudo estar junto e executado de uma única vez.

No final do dia podemos dizer que em um fluxo de "pipeline" pode haver a execução de um ou mais "Jobs", sequencialmente ou em paralelo, onde o término com sucesso de todos esses "Jobs" implica numa execução com sucesso da "pipeline" como um todo.

Você pode consultar "Jobs" executados de um pipeline no Gitlab acessando o menu à esquerda no item "CI/CD" e sub-menu "Jobs".



No exemplo acima eu só tenho um único “Job” que se chama “deploy”, à frente do “Job” está a indicação de qual “pipeline” ele pertence. Se você tem vários “Jobs” na pipeline e uma delas dá erro (por qualquer motivo), a “pipeline” fica como erro também, se você tiver “Jobs” sequenciais, os próximos “Jobs” que seriam executados logo após o “Job” que deu erro não são executados. Isso poupa processamento e tempo gasto do “runner”, o que é excelente para evitar desperdício e ter uma resposta rápida para sabermos o que temos que corrigir.

Outro ponto importante: esses robôs compartilhados estão no ambiente do Gitlab, portanto estão na internet e fora do seu computador, aí vem a pergunta: se eu quiser executar uma pipeline no Gitlab, que está na internet, como é que eu faço para ele chamar o Docker que está no meu computador?

A resposta é: não dá!

Tá legal, mas então como eu resolvo isso?

Crie o seu “runner”!

O Gitlab nos permite usar os “runners” compartilhados do próprio Gitlab mas também podemos criar os nossos! Funciona assim: nós iremos criar um “runner” como se fosse mais um container no nosso Docker e iremos registrar esse “runner” no nosso repositório do Gitlab.

Esse registro é necessário para que o “runner” saiba onde ele tem que olhar para puxar a tarefa para si e rodar o “pipeline”. Esse é o fluxo dos “runners”, não é o Gitlab quem chama o “runner”, na verdade é o “runner” quem fica consultando o Gitlab de tempos em tempos, no repositório à qual está registrado, para verificar se houve alguma mudança no repositório que necessite dar start nos “Jobs” da “pipeline”.

Para criar um “runner”, você poderá utilizar esta estrutura de comando:

docker run -d —name [nome do container] -v [pasta do config do runner]:/etc/gitlab-runner -v /var/run/docker.sock:/var/run/docker.sock gitlab/gitlab-runner:latest

Sendo:

“docker run”: para criar o container e já deixa-lo em execução

“-d”: para que o container suba executando em background

“—name [nome do container]”: para dar um nome ao container, troque o [nome do container] pelo nome que você desejar

“-v [pasta do config do runner]:/etc/gitlab-runner]”: faz um mount de um volume apontando para a pasta onde ficará o arquivo de configuração do “runner”

“-v /var/run/docker.sock:/var/run/docker.sock”: esta é uma das partes mais importantes, aqui compartilha-se o arquivo interno do Docker que dá acesso ao “host” do seu computador, é com isso que conseguimos fazer este “runner” criar os containers temporários auxiliares para executar a pipeline (ou seja, administrar o Docker)

“gitlab/gitlab-runner:latest”: essa é a imagem que utilizaremos para subir o container do Gitlab “runner”

Aqui em minha máquina eu criei meu runner com o seguinte comando:

docker run -d —name gitlab-runner -v /D/docker/volumes/gitlab-runner/config:/etc/gitlab-runner -v /var/run/docker.sock:/var/run/docker.sock gitlab/gitlab-runner:latest

O container do “runner” subirá, mas ele ainda não “observa” nenhum repositório específico, para isso nós temos que executar um comando que cria um container somente para registar o “runner” em um repositório. O comando tem essa sintaxe:

docker run --rm -t -i -v [pasta do config do runner]:/etc/gitlab-runner gitlab/gitlab-runner register --non-interactive --executor “docker” —docker-image “docker:19.03.12” —docker-volumes /var/run/docker.sock:/var/run/docker.sock --url “https://gitlab.com/” --registration-token “[token do repositório]” —description “docker-runner” --run-untagged=“true” —locked=“false”

Sendo:

“docker run”: para criar o container e já deixa-lo em execução

“—rm”: automaticamente destrói o container após execução de todas as etapas

“-t”: para verificar se o meio de saída é um terminal para executar comandos

“-i”: mantém ativo uma entrada para comandos (como se houvesse um teclado plugado)

“-v [pasta do config do runner]:/etc/gitlab-runner”: necessário o mount para a mesma pasta de configuração do “runner”, após o registro informações desse registro são atualizadas no arquivo de configuração, por isso o “runner” que subimos anteriormente saberá para onde olhar

“gitlab/gitlab-runner”: é o nome da imagem a ser utilizada

“register”: é um subcomando passado à imagem, será chamado na subida do container

“—non-interactive”: indica para não registrar abrindo o terminal no modo interativo (para não precisar abrir tela para digitação)

“–executor “docker””: indica o tipo de executor que o “runner” usará para executar os comandos do script a cada execução de pipeline

“–docker-image “docker:19.03.12””: indica a imagem base a ser utilizada ao pelo Docker do “runner” no momento de execução da pipeline

“-v /var/run/docker.sock:/var/run/docker.sock”: esta é uma das partes mais importantes, aqui compartilha-se o arquivo interno do Docker que dá acesso ao “host” do seu computador com o container gerado pelo “runner” em momento de execução, é com isso que conseguimos fazer este container temporário gerado pelo “runner” criar imagens e containers (ou seja, administrar o Docker)

“–url https://gitlab.com/": indica onde será feito o registro, no caso é o próprio site do Gitlab

“–registration-token “[token do repositório]””: é aqui que você indica o token do seu repositório, é assim que o registrador sabe qual repositório do Gitlab que será feito o registro

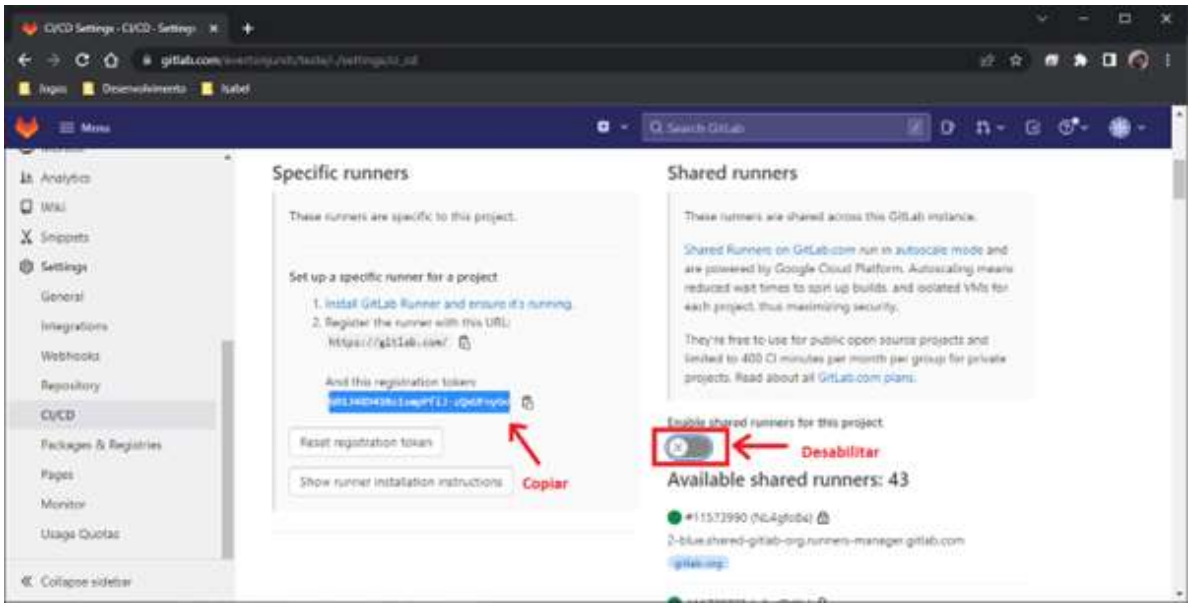
“–description “docker-runner””: é uma descrição amigável que vai aparecer no site do Gitlab para você saber que é o “runner” que você está registrando. Se você for registrar mais “runners”, aconselha-se nomear com um nome diferente de um que já esteja registrado

“–run-untagged=“true””: desobriga a existência de uma “tag” (etiqueta) no “Job” para que o “runner” execute a pipeline

“–locked=“false””: desobriga o “runner” a ser usado em projetos específicos, isso é bom no nosso caso em que usaremos o mesmo “runner” em alguns projetos diferentes mais pra frente.

Antes de executar o comando, você tem que copiar o token que está no seu repositório. No menu à esquerda em “Settings”, sub-menu “CI/CD”, vá na guia “Runners” e clique no botão “Expand”, primeiro desabilite o item “Enable shared

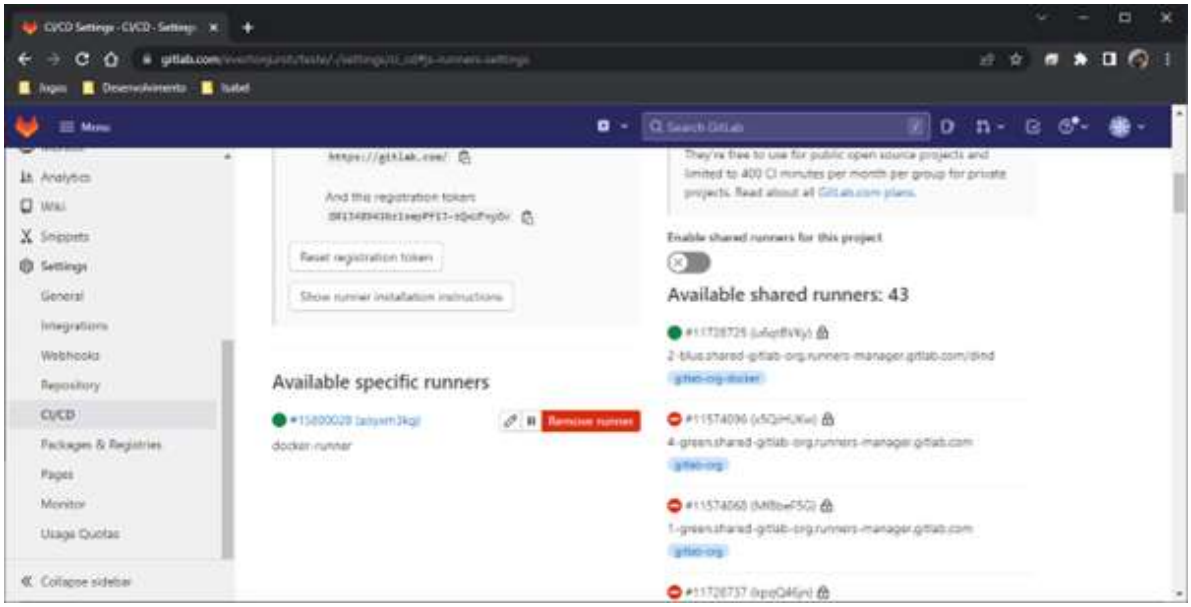
runners for this project" para não usar nenhum "runner" compartilhado, já que os mesmos não chegarão até o seu computador, e depois copie o "registration token", conforme indicado na figura à seguir:



O comando abaixo é um exemplo que eu usei para registrar o token do meu repositório, lembre-se de nunca compartilhar seu token, no meu caso depois eu fiz o "Reset registration token" que troca o token por um novo.

docker run --rm -t -i -v /D/docker/volumes/gitlab-runner/config:/etc/gitlab-runner gitlab/gitlab-runner register —non-interactive --executor "docker" —docker-image "docker:19.03.12" —docker-volumes /var/run/docker.sock:/var/run/docker.sock --url "https://gitlab.com/" --registration-token "GR1348941Nz1smpPfiJ-zQxUFnyGv" —description "docker-runner" —run-untagged="true" --locked="false"

Para ver se tudo deu certo, volte à página do Gitlab e atualize a página (refresh — tecla F5), o nosso "runner" aparecerá do lado esquerdo, abaixo de onde você pegou o token:



Bom, vamos testar!!!

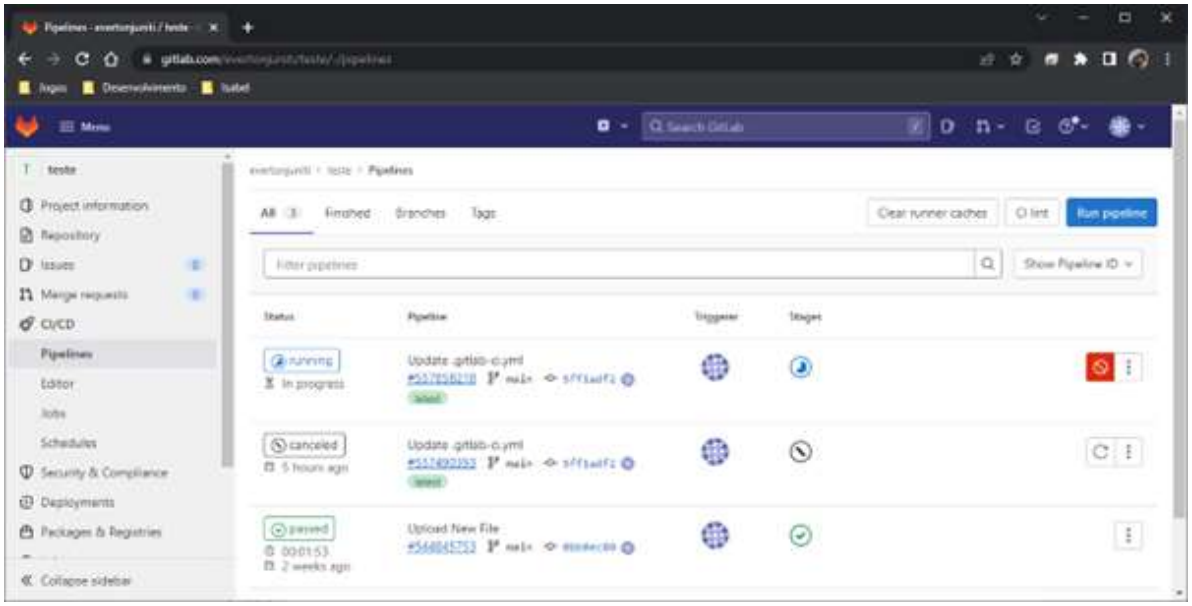
Crie um arquivo com o nome “.gitlab-ci.yml” (sem as aspas duplas) com o seguinte conteúdo:

```
stages:

  - deploy

deploy:

  stage: deploy

  script:

    docker run -d —name Nginx_teste -p 3080:80 nginx:latest
```

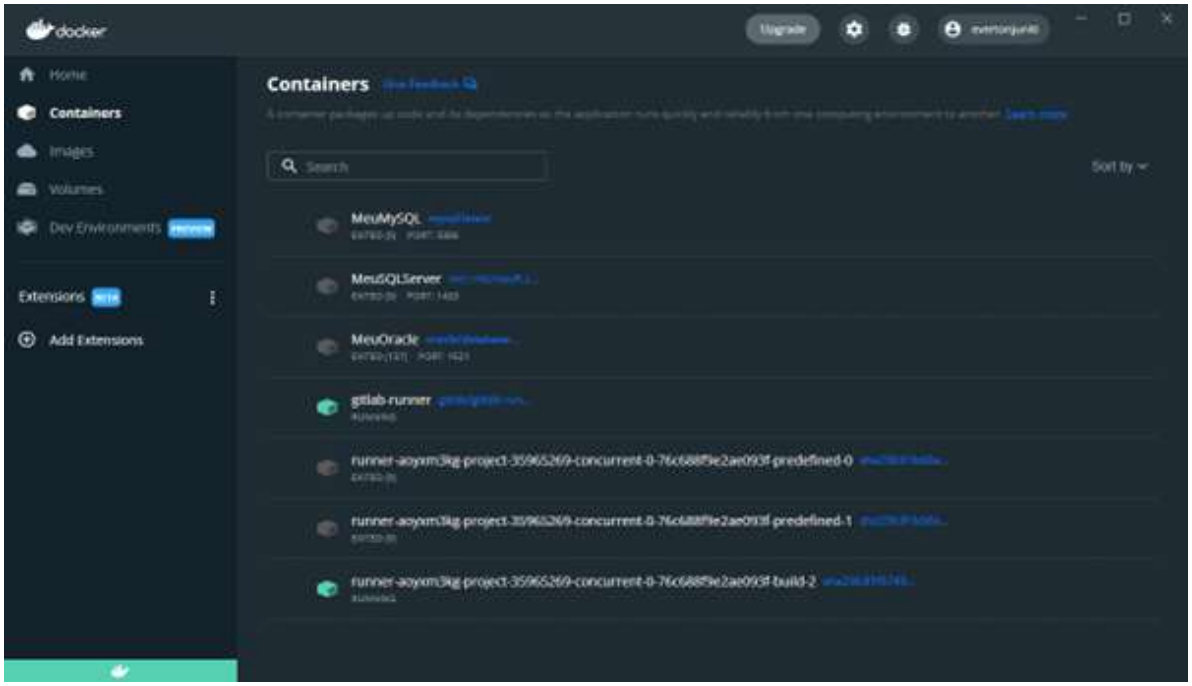
Suba este arquivo “.gitlab-ci.yml” no seu repositório (em qualquer lugar).

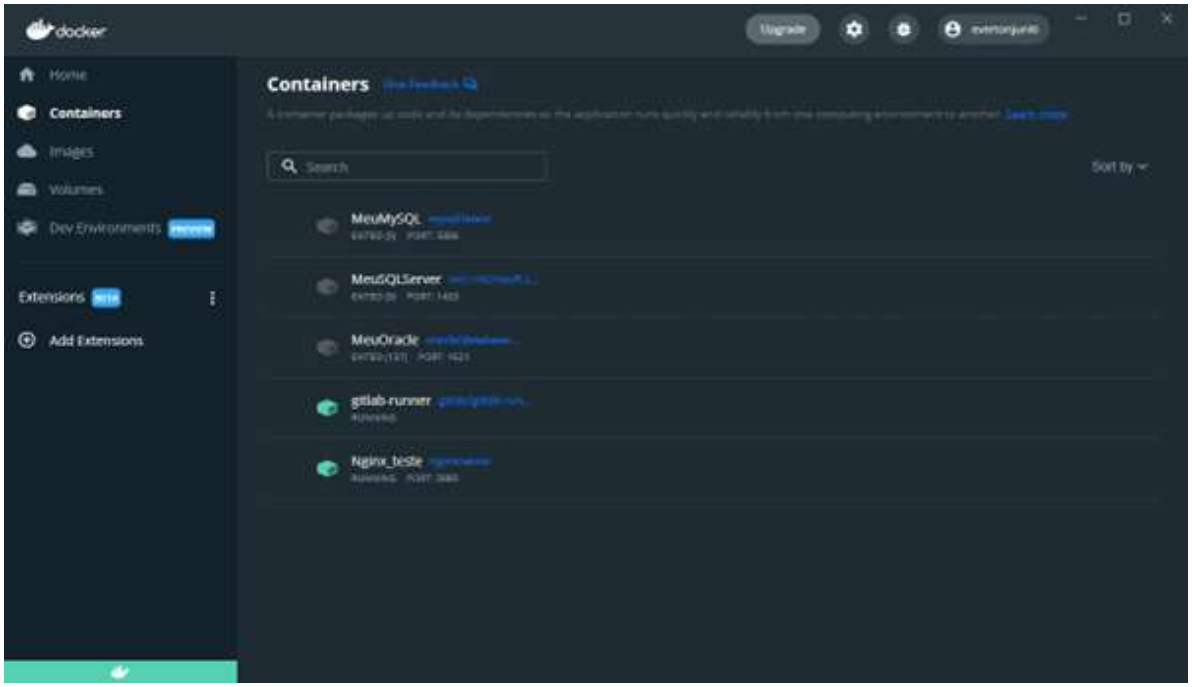
Não se preocupe com o conteúdo, o que você precisa saber é que ele será lido pelo nosso “runner” no Docker da nossa máquina que lerá esta receita. A receita diz para criar um container chamado “Nginx_teste” com um bind na porta “3080” e com a imagem “nginx:latest”

Assim que subir o arquivo no repositório, vá no menu à esquerda em “CI/CD” no sub-menu “Pipelines”. Você observará que um pipeline foi iniciado:



Abra o Docker Desktop, você verá que várias coisas estarão acontecendo, é o “runner” criando containers temporários para rodar o Docker e executar o script da pipeline, quando terminar você verá um container novo chamado “Nginx_teste” criado:





Atividade Extra

Para se aprofundar no assunto desta aula leia a documentação oficial: “Gitlab Runner”.

Link da documentação: https://docs.gitlab.com/runner/

Outra documentação oficial complementar é o: “Gitlab CI/CD”

Link da documentação: https://docs.gitlab.com/ee/ci/

Referência Bibliográfica

GitLab CI/CD. Disponível em: https://docs.gitlab.com/ee/ci/. Acesso em: 16 de junho de 2022.

Ir para exercício