Subindo banco de dados via pipeline

Muito bem! Chegamos ao momento mágico de finalmente criarmos nossas pipelines apartadas (fora do repositório de teste) para subirmos nossa solução por completo!

Iremos começar com os bancos de dados, temos 2 para subirmos: Oracle e MySQL.

Vamos começar com o MySQL pois ele é menor e versátil para fins de aprendizado.

No Gitlab, crie um repositório novo, pode ser um projeto em branco mesmo, se possível com o nome "mysql" 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!

Usaremos o script_inicial.sql e o script_tabelas.sql após subir nosso container pela primeira vez!

Eu sei que subimos lá atrás um banco de dados MySQL, mas faremos tudo do zero aqui via pipeline, então vá no Docker Desktop e **exclua** o container que criamos chamado "MeuMySQL". A imagem do mysql:latest você poderá manter, se você excluir a única diferença é que haverá o download da imagem quando subirmos o

pipeline, se a imagem já estiver lá pulamos esta etapa, aí é com você a decisão de excluir a imagem ou manter! Como estamos zerando tudo, se você fez como eu a criação de um volume, delete todo o conteúdo do volume para criarmos do zero mesmo! No meu caso eu excluí todo o conteúdo da pasta D:\docker\volumes\mysql que é a que existe no meu computador.

Vamos lembrar de uma coisa muito importante, o comando que usamos para subir o container:

```
docker run --name MeuMySQL --network MinhaRede -v /D/docker/volumes/mysql:/var/lib/mysql -p 3306:3306 -e MYSQL_ROOT_PASSWORD=Minha@senha -e MYSQL_USER=MeuUsuario -e MYSQL_PASSWORD=MinhaSenha -d mysql:latest
```

Tem algumas informações que não são muito legais mantermos aberto no script de pipeline de nosso repositório, veremos mais à frente como guardar isso no Gitlab para não precisarmos deixar explícito assim no script, por enquanto apenas tome nota.

Pronto! Temos um repositório preparado, agora vamos para o outro:

Faça a criação de outro repositório (sim, tem que estar separado, pois só pode haver 1 pipeline, ou seja 1 .gitlab-ci.yml, por repositório), agora chamado de oracle (veja se você já não possui um projeto com este nome em seu repositório, se tiver escolha outro nome). Importante você criar isso no **SEU** repositório hein!

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

Teremos que fazer a cópia de um arquivo que está no repositório https://gitlab.com/everton.juniti/descomplica, na pasta cicd_database\oracle, somente o arquivo script_inicial.sql. Crie uma pasta chamada setup no diretório do Windows você criou para o oracle e coloque esse arquivo script_inicial.sql dentro dessa pasta setup, no meu caso está em

D:\docker\volumes\oracle\setup. Isso será necessário para configurarmos o volume em que o container lerá o script inicial.

Eu sei que subimos lá atrás um banco de dados Oracle também, mas faremos tudo do zero aqui via pipeline, então vá no Docker Desktop e **exclua** o container que criamos chamado "MeuOracle". A imagem do oracle/database:18.4.0-xe você **não poderá manter**, lembre que no caso do Oracle nós fizemos o build de uma imagem e aqui teremos que fazer novamente, então **exclua também a imagem**! Como estamos zerando tudo, se você fez como eu a criação de um volume, delete todo o conteúdo do volume para criarmos do zero mesmo! No meu caso eu excluí todo o conteúdo da pasta D:\docker\volumes\oracle que é a que existe no meu computador.

Vamos lembrar de outra coisa muito importante, o comando que usamos para subir o container:

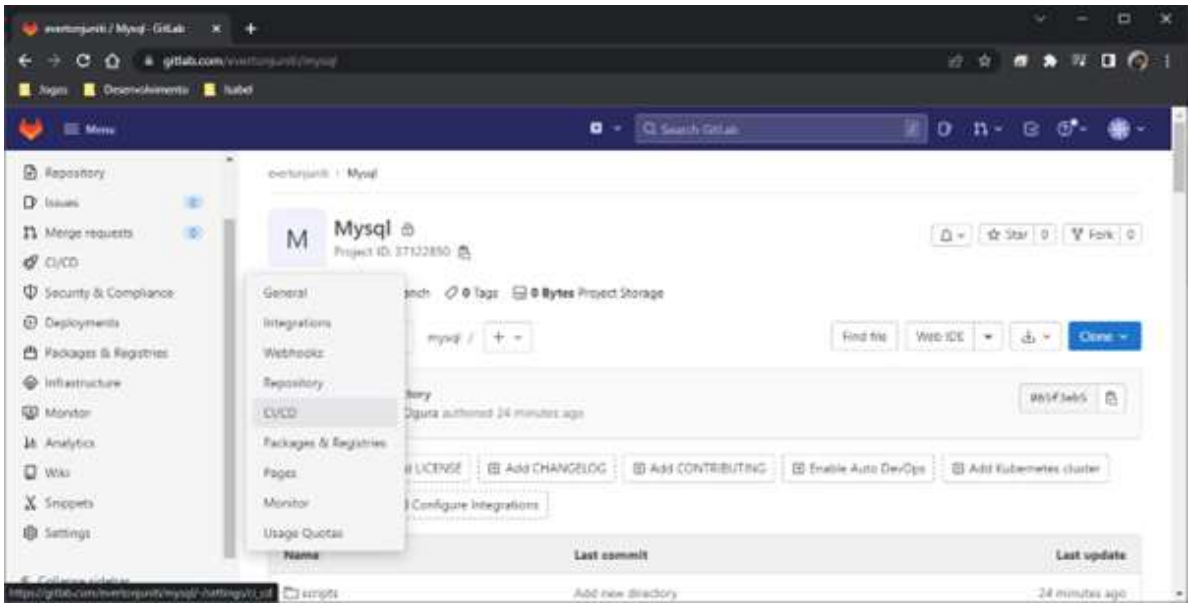
```
docker run --name MeuOracle --network MinhaRede -v /D/docker/volumes/oracle/oradata:/opt/oracle/oradata -v /D/docker/volumes/oracle/setup:/opt/oracle/scripts/setup -p 1521:1521 -p 5500:5500 -e ORACLE_PWD=MinhaSenha -e ORACLE_CHARACTERSET=AL32UTF8 -d oracle/database:18.4.0-xe
```

Também é uma informação que não é muito legal mantermos aberto no script de pipeline de nosso repositório assim como no exemplo do MySQL, veremos mais à frente como guardar isso no Gitlab para não precisarmos deixar explícito assim no script, lembre-se das duas anotações, a do MySQL e a do Oracle a respeito de senhas!

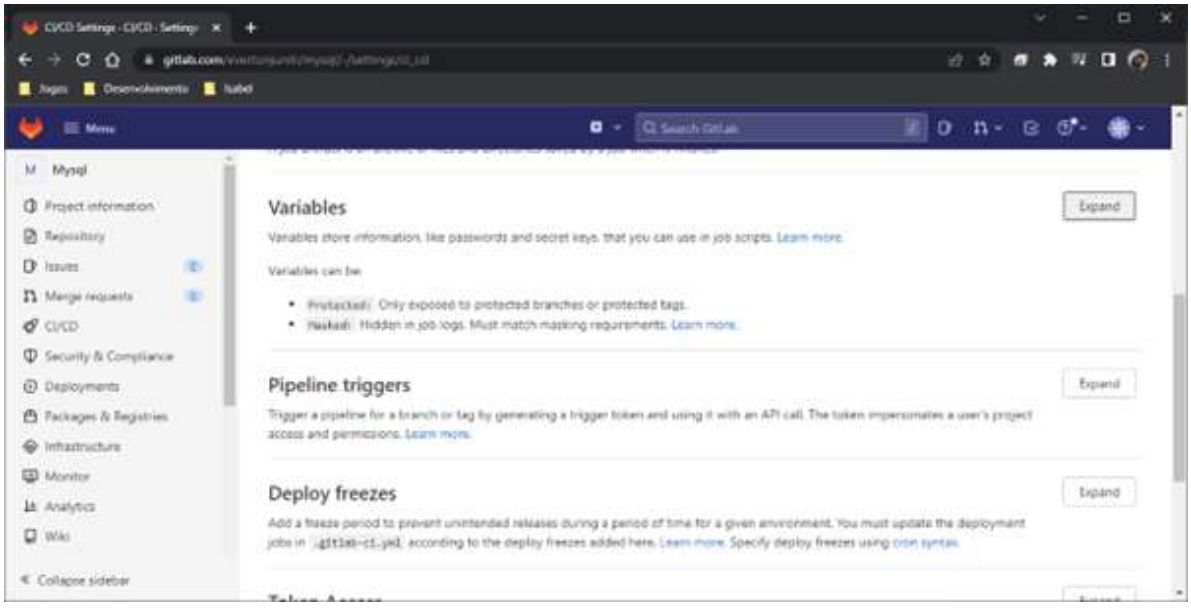
Tem algumas maneiras de se gravarem o que chamamos de Secrets (segredos), na documentação de referência tem até instruções para uso de fontes externas de gestão de senhas (https://docs.gitlab.com/ee/ci/secrets/), mas aqui faremos algo um pouco mais simples, até porque criar um cofre externo neste caso não seria muito didático, o importante aqui é saber que é possível não expor senhas de forma aberta.

Para isso você terá que ir em cada repositório e criar algumas variáveis do Gitlab, vamos fazer primeiro no repositório do MySQL.

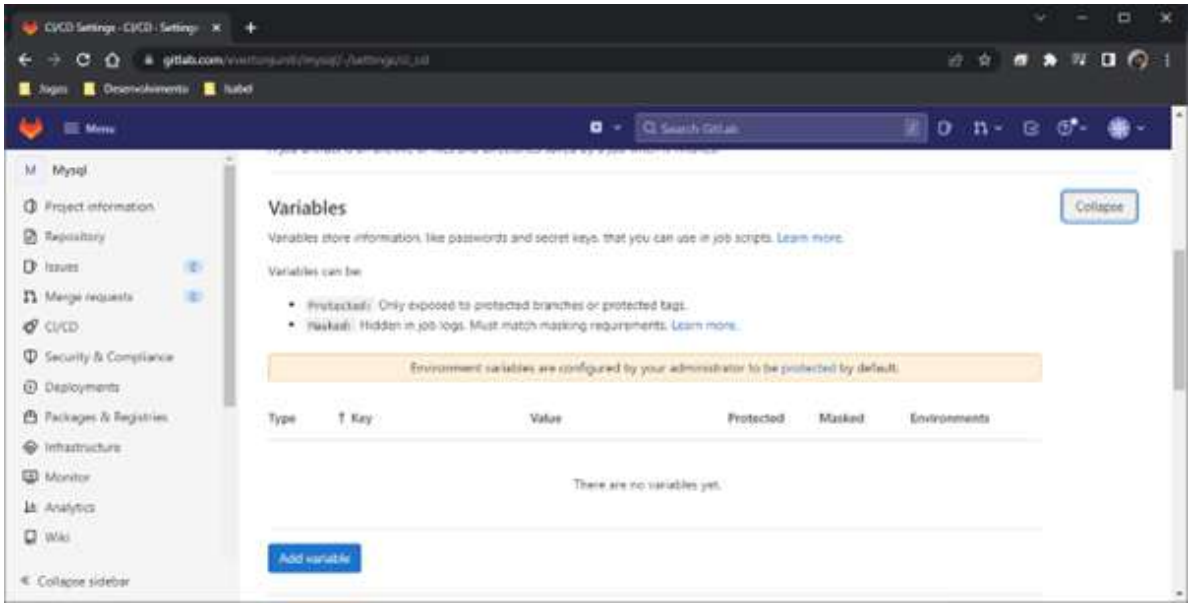
No repositório do MySQL, vá no menu lateral à esquerda no item “Settings” e depois clique em “CI/CD”.



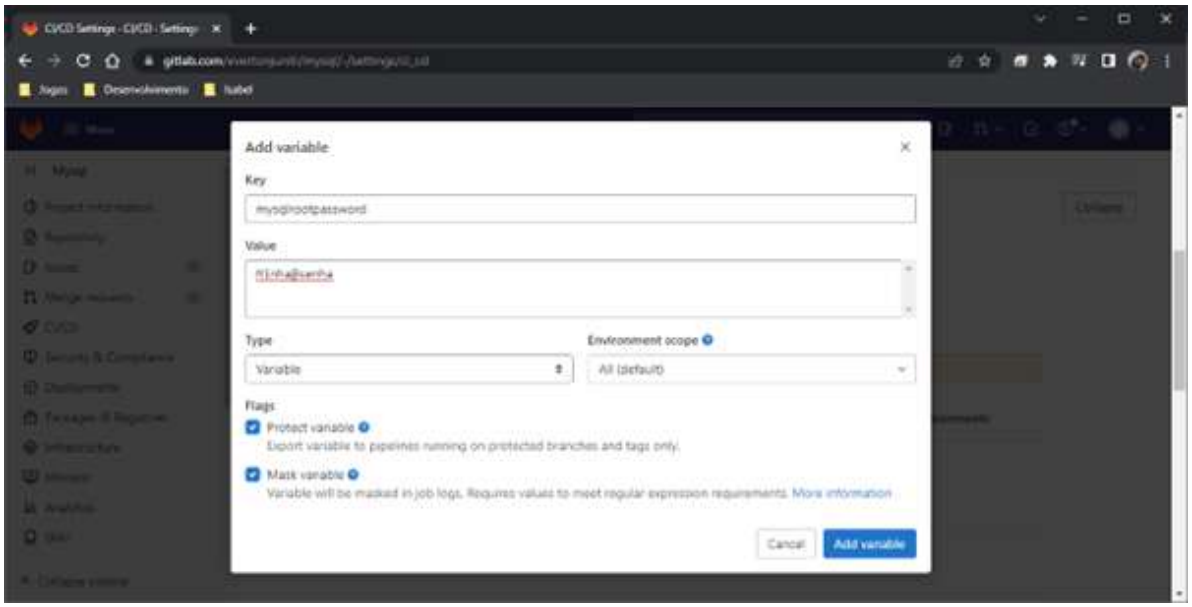
Clique no botão “Expand” na guia “Variables”.

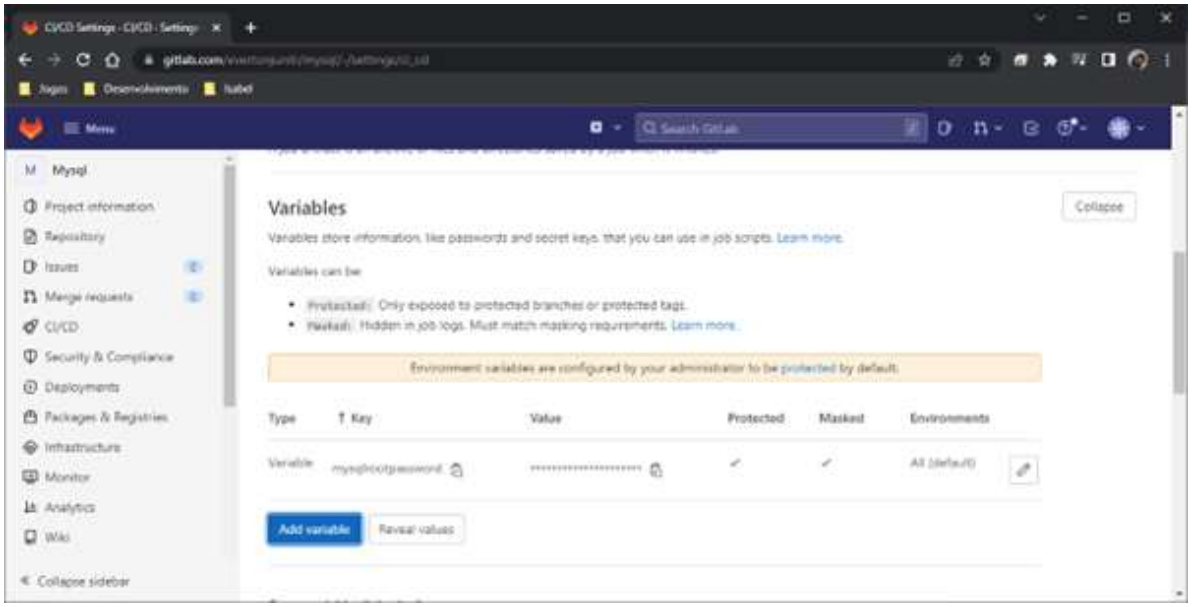


Clique no botão “Add Variable”, vamos criar algumas variáveis para o usuário root do MySQL, o usuário “normal” e a senha do usuário “normal”.



Neste primeiro momento vamos criar uma variável chamada mysqlrootpassword com o valor Minha@senha, marque a opção “Mask variable”, o restante deixe o padrão. Depois clique no botão “Add variable”.



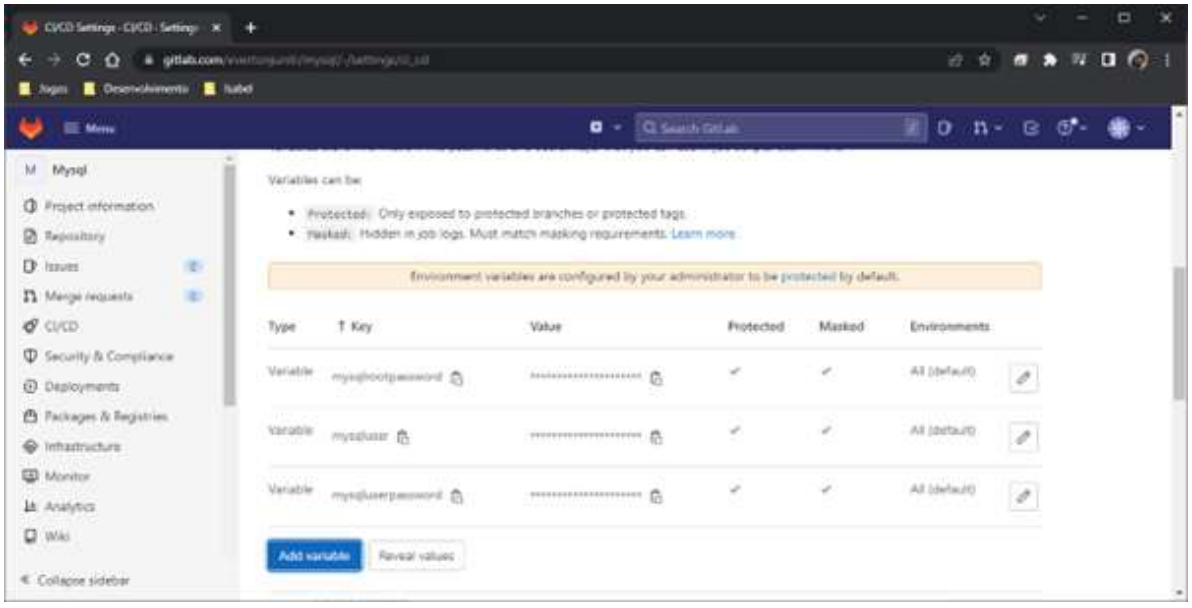


Faça o mesmo procedimento para criar mais 2 variáveis, com a opção de mascaramento também:

Variável mysqluser, valor MeuUsuario

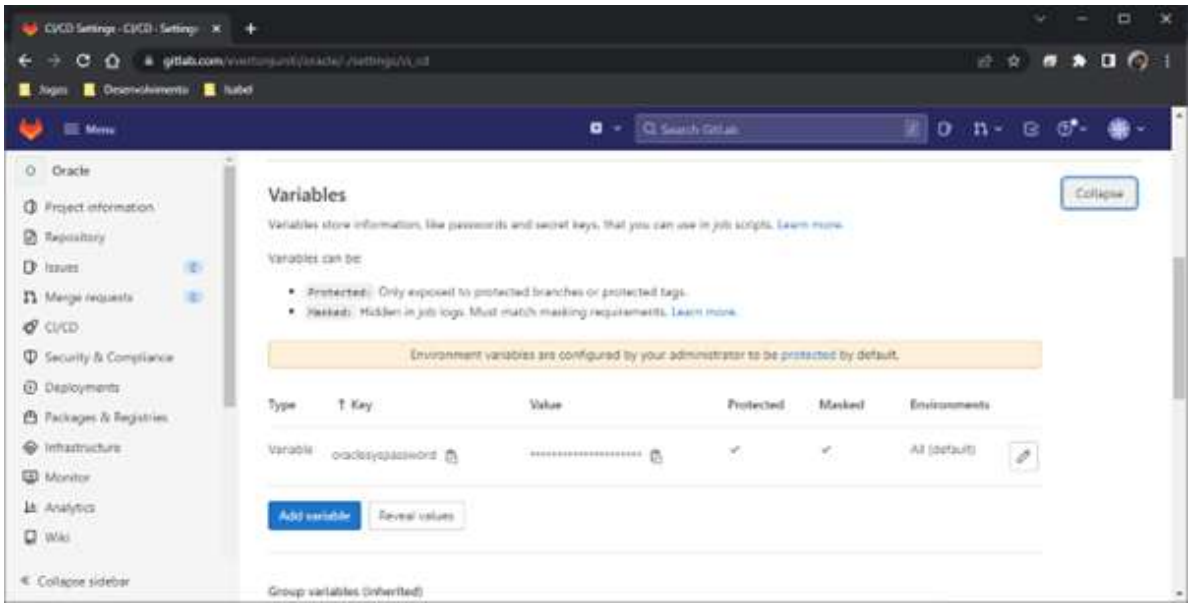
Variável mysqluserpassword, valor MinhaSenha

Ao final deverá ficar algo assim:



No repositório do Oracle vamos fazer a mesma coisa, criando a seguinte variável:

Variável oraclesyspassword, valor MinhaSenha



Agora vamos voltar ao repositório https://gitlab.com/everton.juniti/descomplica para olharmos o script .gitlab-ci.yml da pasta cicd_database\mysql\01-Before_After_script.

```
variables:
    IMAGEM_MYSQL: mysql:latest
    BIND_DE_PORTA: 3306:3306
    PASTA_ORIGEM_VOLUME: /D/docker/volumes/mysql
    NOME_DA_REDE: MinhaRede
    NOME_DO_CONTAINER: MeuMySQL
    CONTAINER_ATIVO: falso
    CONTAINER_EXISTE: falso

stages:
    - deploy
    - start
    - scripts

before_script:
    - RUNNING_CONTAINER_ID=$(docker container ls -f name=$NOME_DO_CONTAINER -f
status=running -q)
    - if [[ -z $RUNNING_CONTAINER_ID ]]; then
    -   CONTAINER_ATIVO=falso;
    -   else CONTAINER_ATIVO=verdadeiro;CONTAINER_EXISTE=verdadeiro;
    - fi
    - if [[ $CONTAINER_ATIVO = "falso" ]]; then
    -   CONTAINER_ID=$(docker container ls -f name=$NOME_DO_CONTAINER -a -q)
    -   if [[ -z $CONTAINER_ID ]]; then
    -     CONTAINER_EXISTE=falso
    -     else CONTAINER_EXISTE=verdadeiro
    -   fi
```

```
    - fi

Implantacao_MySQL:
    stage: deploy
    variables:
        GIT_STRATEGY: none
    script:
        - if [[ $CONTAINER_EXISTE = "falso" ]]; then
        -   docker run --name $NOME_DO_CONTAINER --network $NOME_DA_REDE -v
$PASTA_ORIGEM_VOLUME:/var/lib/mysql -p $BIND_DE_PORTA -e
MYSQL_ROOT_PASSWORD=$mysqlrootpassword -e MYSQL_USER=$mysqluser -e
MYSQL_PASSWORD=$mysqluserpassword -d $IMAGEM_MYSQL
        -   else echo "Container não precisa ser criado"
        - fi

Ligar_Container:
    stage: start
    variables:
        GIT_STRATEGY: none
    script:
        - if [[ -z $RUNNING_CONTAINER_ID ]]; then
        -   CONTAINER_ID=$(docker container ls -f name=$NOME_DO_CONTAINER -a -
q)
        -   if [[ -z $CONTAINER_ID ]]; then
        -     CONTAINER_EXISTE=falso
        -     else CONTAINER_EXISTE=verdadeiro
        -   fi
        -   if [[ $CONTAINER_EXISTE = "verdadeiro" ]]; then
        -     docker start $NOME_DO_CONTAINER
        -     RUNNING_CONTAINER_ID=$(docker container ls -f
name=$NOME_DO_CONTAINER -f status=running -q)
        -   fi
        -   else echo "Container não precisa ser iniciado"
        - fi

after_script:
    - RUNNING_CONTAINER_ID=$(docker container ls -f name=$NOME_DO_CONTAINER -f
status=running -q)
    - if [[ -n $RUNNING_CONTAINER_ID ]]; then
    -   echo "Id do Container=$RUNNING_CONTAINER_ID"
    -   else "Sem container ativo"
    - fi
```

Aqui a novidade é o before_script e o after_script. Do jeito que foram declarados, estão como algo geral em todo o script de pipeline.

O que estiver no before_script será executado em todos os jobs antes do que estiver em **script.**

E o que estiver no after_script será executado em todos os jobs depois do que estiver em **script.**

Se você colocar tanto o before_script ou o after_script dentro de um job, o que estiver dentro do job irá sobrepor o before e after geral.

Este script tem um conjunto de verificações, se o container existir e estiver rodando nada será feito, se o container não estiver rodando mas existir então o script tentará iniciar o container, agora se o container não existir então será feita a tentativa de criar o container.

Agora vamos voltar ao repositório https://gitlab.com/everton.juniti/descomplica para olharmos o script .gitlab-ci.yml da pasta cicd_database\mysql\02-Ony_Except_script.

```
variables:
    IMAGEM_MYSQL: mysql:latest
    BIND_DE_PORTA: 3306:3306
    PASTA_ORIGEM_VOLUME: /D/docker/volumes/mysql
    NOME_DA_REDE: MinhaRede
    NOME_DO_CONTAINER: MeuMySQL
    CONTAINER_ATIVO: falso
    CONTAINER_EXISTE: falso

stages:
    - deploy
    - start
    - scripts

before_script:
    - RUNNING_CONTAINER_ID=$(docker container ls -f name=$NOME_DO_CONTAINER -f
status=running -q)
    - if [[ -z $RUNNING_CONTAINER_ID ]]; then
    -   CONTAINER_ATIVO=falso;
    -   else CONTAINER_ATIVO=verdadeiro;CONTAINER_EXISTE=verdadeiro;
    - fi
    - if [[ $CONTAINER_ATIVO = "falso" ]]; then
    -   CONTAINER_ID=$(docker container ls -f name=$NOME_DO_CONTAINER -a -q)
    -   if [[ -z $CONTAINER_ID ]]; then
    -     CONTAINER_EXISTE=falso
    -     else CONTAINER_EXISTE=verdadeiro
    -   fi
    - fi

Implantacao_MySQL:
    stage: deploy
    only:
      - /^release_[0-9]+(?:.[0-9]+)+$/
```

```
    variables:
        GIT_STRATEGY: none
    script:
        - if [[ $CONTAINER_EXISTE = "falso" ]]; then
        -   docker run --name $NOME_DO_CONTAINER --network $NOME_DA_REDE -v
$PASTA_ORIGEM_VOLUME:/var/lib/mysql -p $BIND_DE_PORTA -e
MYSQL_ROOT_PASSWORD=$mysqlrootpassword -e MYSQL_USER=$mysqluser -e
MYSQL_PASSWORD=$mysqluserpassword -d $IMAGEM_MYSQL
        -   else echo "Container não precisa ser criado"
        - fi

Ligar_Container:
    stage: start
    except:
      - main
    variables:
        GIT_STRATEGY: none
    script:
        - if [[ -z $RUNNING_CONTAINER_ID ]]; then
        -   CONTAINER_ID=$(docker container ls -f name=$NOME_DO_CONTAINER -a -
q)
        -   if [[ -z $CONTAINER_ID ]]; then
        -     CONTAINER_EXISTE=falso
        -     else CONTAINER_EXISTE=verdadeiro
        -   fi
        -   if [[ $CONTAINER_EXISTE = "verdadeiro" ]]; then
        -     docker start $NOME_DO_CONTAINER
        -     RUNNING_CONTAINER_ID=$(docker container ls -f
name=$NOME_DO_CONTAINER -f status=running -q)
        -   fi
        -   else echo "Container não precisa ser iniciado"
        - fi

after_script:
    - RUNNING_CONTAINER_ID=$(docker container ls -f name=$NOME_DO_CONTAINER -f
status=running -q)
    - if [[ -n $RUNNING_CONTAINER_ID ]]; then
    -   echo "Id do Container=$RUNNING_CONTAINER_ID"
    -   else "Sem container ativo"
    - fi
```

Vejamos a sintaxe do only:

```
  only:
    - /^release_[0-9]+(?:.[0-9]+)+$/
```

Ao indicar a instrução only no job, indicamos que o job só poderá ser executado sob as circunstâncias indicadas no only, no nosso caso somente quando houver uma alteração em uma branch iniciada com "release_x.y", onde x e y são números, ou seja, se criarmos uma branch chamada release_1.0, o pipeline será iniciado para execução do job "Implantacao_MySQL".

Agora vamos ver o except:

```
  except:
    - main
```

Aqui a intenção é oposta, o job com a instrução except sempre executará exceto sob a circunstância indicada no except, no nosso caso quando houver uma alteração na branch main, o job não será executado.

Agora vamos voltar ao repositório https://gitlab.com/everton.juniti/descomplica para olharmos o script .gitlab-ci.yml da pasta cicd_database\oracle.

```yaml
variables:
    IMAGEM_ORACLE: oracle/database:18.4.0-xe
    BIND_DE_PORTA: 3306:3306
    PASTA_ORIGEM_VOLUME: /D/docker/volumes/oracle/oradata
    PASTA_ORIGEM_VOLUME_SETUP: /D/docker/volumes/oracle/setup
    NOME_DA_REDE: MinhaRede
    NOME_DO_CONTAINER: MeuOracle
    CONTAINER_ATIVO: falso
    CONTAINER_EXISTE: falso

stages:
    - build
    - deploy
    - start

before_script:
    - RUNNING_CONTAINER_ID=$(docker container ls -f name=$NOME_DO_CONTAINER -f
status=running -q)
    - if [[ -z $RUNNING_CONTAINER_ID ]]; then
    -   CONTAINER_ATIVO=falso;
    -   else CONTAINER_ATIVO=verdadeiro;CONTAINER_EXISTE=verdadeiro;
    - fi
    - if [[ $CONTAINER_ATIVO = "falso" ]]; then
    -   CONTAINER_ID=$(docker container ls -f name=$NOME_DO_CONTAINER -a -q)
    -   if [[ -z $CONTAINER_ID ]]; then
    -     CONTAINER_EXISTE=falso
    -     else CONTAINER_EXISTE=verdadeiro
    -   fi
    - fi

Construir_Oracle:
    stage: build
    only:
      - /^release_[0-9]+(?:.[0-9]+)+$/
    before_script:
      - IMAGE_ID=$(docker image ls --filter=reference=$IMAGEM_ORACLE -q)
      - if [[ -z $IMAGE_ID ]]; then
          IMAGEM_EXISTE=falso
      - fi
    script:
      - if [[ $IMAGEM_EXISTE = "falso" ]]; then
      -   apk update && apk add git
      -   git clone https://github.com/oracle/docker-images.git
      -   cd ./docker-images/OracleDatabase/SingleInstance/dockerfiles
      -   apk add bash
      -   bash buildContainerImage.sh -v 18.4.0 -x
      - fi

Implantacao_Oracle:
```

```
    stage: deploy
    only:
      - /^release_[0-9]+(?:.[0-9]+)+$/
    variables:
        GIT_STRATEGY: none
    script:
        - if [[ $CONTAINER_EXISTE = "falso" ]]; then
        -   docker run --name $NOME_DO_CONTAINER --network $NOME_DA_REDE -v
$PASTA_ORIGEM_VOLUME:/opt/oracle/oradata -v
$PASTA_ORIGEM_VOLUME_SETUP:/opt/oracle/scripts/setup -p 1521:1521 -p 5500:5500
-e ORACLE_PWD=$oraclesyspassword -e ORACLE_CHARACTERSET=AL32UTF8 -d
$IMAGEM_ORACLE
        -   else echo "Container não precisa ser criado"
        - fi

Ligar_Container:
    stage: start
    except:
      - main
    variables:
        GIT_STRATEGY: none
    script:
        - if [[ -z $RUNNING_CONTAINER_ID ]]; then
        -   CONTAINER_ID=$(docker container ls -f name=$NOME_DO_CONTAINER -a -
q)
        -   if [[ -z $CONTAINER_ID ]]; then
        -     CONTAINER_EXISTE=falso
        -     else CONTAINER_EXISTE=verdadeiro
        -   fi
        -   if [[ $CONTAINER_EXISTE = "verdadeiro" ]]; then
        -     docker start $NOME_DO_CONTAINER
        -     RUNNING_CONTAINER_ID=$(docker container ls -f
name=$NOME_DO_CONTAINER -f status=running -q)
        -   fi
        -   else echo "Container não precisa ser iniciado"
        - fi

after_script:
    - RUNNING_CONTAINER_ID=$(docker container ls -f name=$NOME_DO_CONTAINER -f
status=running -q)
    - if [[ -n $RUNNING_CONTAINER_ID ]]; then
    -   echo "Id do Container=$RUNNING_CONTAINER_ID"
    -   else "Sem container ativo"
    - fi
```

Aqui a diferença em relação ao script do MySQL é a adição de um **stage** de build, uma vez que a imagem não está disponível publicamente e teremos que efetuar o build da imagem do Oracle antes de tentar criar um container.

Muito bem, vamos subir os 2 scripts em seus respectivos repositórios!

No seu repositório do MySQL coloque uma cópia do arquivo .gitlab-ci.yml que se encontra no repositório https://gitlab.com/everton.juniti/descomplica, na pasta cicd_database\mysql.

No seu repositório do Oracle coloque uma cópia do arquivo .gitlab-ci.yml que se encontra no repositório https://gitlab.com/everton.juniti/descomplica, na pasta cicd_database\oracle.

Em ambos os repositórios, vá no menu lateral à esquerda em Settings e depois clique em CI/CD.

Clique no botão “Expand” na guia “Runners”.

Aquele runner que utilizamos no repositório de testes aparecerá, mas você precisa clicar no botão “Enable for this project” para habilitar aquele nosso runner para executar as pipelines, faça isso nos 2 repositórios (do MySQL e do Oracle).

Como a criação da imagem do Oracle demora bastante, sugiro você clicar no botão “Expand” da guia “General pipelines” (ainda em Settings -> CI/CD) para aumentar o valor do campo “Timeout” de “1h” para “2h”, significa que o tempo total de 2 horas será concedido à execução da pipeline, caso atinja as 2h e a pipeline não consiga ser executada por completo, a pipeline falhará.

Pronto! Agora sua pipeline está pronta para criar os 2 containers, como configuramos via only e except para rodar os jobs somente por uma branch que comece com “release”, você poderá criar uma branch chamada “release_1.0” (sem as aspas duplas) tomando como base a branch main, assim que você fizer isso sua pipeline se iniciará!

Observação importante: como indicamos nas variáveis que criamos (as secrets) que elas só poderiam ser utilizadas em branches protegidas ou com tags, você terá que proteger essas branches chamadas releases que criar! Assim que criar, vá no menu lateral “Settings” e clique em “Repository”, clique no botão “Expand” na guia “Protected branches” e selecione a branch de release que criou para protege-la, marcando as opções “Allowed to merge” com a opção “Developers + Maintainers” (ou só “Maintainers”) e a opção “Allowed to push” como “None” (forçando ter que fazer merge request de outra branch para esta).

Caso você venha a fazer modificações, você poderá criar outras branches (por exemplo release_1.1) ou criar novas branches à partir da main. O importante é que a pipeline só será iniciada à partir dessas branches que começam com “release_” ao invés de iniciarem com a branch main.

Você poderá configurar sua pipeline para mudar isso, caso deseje.

Atividade Extra

Para se aprofundar no assunto desta aula leia o documento de referência: “Mask a CI/CD variable”.

Neste documento há as regras para permitir o mascaramento de valores de variáveis.

Link do documento: https://docs.gitlab.com/ee/ci/variables/index.html#mask-a-cicd-variable

No documento de referência “.gitlab-ci.yml keyword reference” você encontra um pouco mais de detalhes sobre o before_script e after_script.

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

No documento de referência “Choose when to run Jobs” você encontra opções adicionais ao only e except.

Link do documento: https://docs.gitlab.com/ee/ci/jobs/job_control.html

Referência Bibliográfica

- OGURA, Everton J. Repositório do GitLab, projeto Descomplica. Disponível em https://gitlab.com/everton.juniti/descomplica. Acesso em 17 de junho de 2022.
- SCHKN. Bash If Else Syntax With Examples. Disponível em https://devconnected.com/bash-if-else-syntax-with-examples/. Acesso em 17 de junho de 2022
- .gitlab-ci.yml keyword reference. Disponível em https://docs.gitlab.com/ee/ci/yaml/. Acesso em 17 de junho de 2022
- Secrets. Disponível em https://docs.gitlab.com/charts/installation/secrets.html. Acesso em 17 de junho de 2022.
- Using external secrets in CI. Disponível em https://docs.gitlab.com/ee/ci/secrets/. Acesso em 17 de junho de 2022.
- docker start. Disponível em https://docs.docker.com/engine/reference/commandline/start/. Acesso em 17 de junho de 2022
- Choose when to run Jobs. Disponível em https://docs.gitlab.com/ee/ci/jobs/job_control.html. Acesso em 17 de junho de 2022

Ir para exercício