Universidade Federal de Santa Catarina
Bacharelado em Sistemas de Informação

Lucas Estevão de Andrade

Implantação de um Processo de Integração e *Deployment* Contínuos em um Laboratório de Pesquisa

Florianópolis
2022

Lucas Estevão de Andrade

Implantação de um Processo de Integração e Deployment Contínuos em um Laboratório de Pesquisa

Trabalho apresentado no curso de Bacharelado em Sistemas de Informação, da Universidade Federal de Santa Catarina (UFSC).

Professor Orientador:
Jean Carlo Rossa Hauck, Dr

Florianópolis
2022

Resumo

Manter a qualidade da entrega de softwares desenvolvidos é uma tarefa complicada, que tende a aumentar em complexidade conforme o software cresce. Ter uma maneira de automatizar os processos de entrega e instalação pode facilitar bastante o dia a dia de um time de desenvolvimento de software. Fluxos de Integração Contínua (ou *Continuous Integration* - CI) e de *Deployment* Contínuo (ou *Continuous Deployment* - CD) automatizam esses processos de entrega, integração, e até de testes de software. O Grupo de Pesquisas em Qualidade de Software (GQS) do Departamento de Informática e Estatística da Universidade Federal de Santa Catarina desenvolve projetos de software relacionados à Engenharia de Software. Atualmente não existe um processo de Integração e Deployment Contínuos no GQS. Assim, este projeto objetiva analisar as características dos softwares desenvolvidos no GQS, e definir e implantar um processo de integração contínua. Neste trabalho, primeiramente, é identificada a necessidade da implantação deste processo no laboratório, analisando o contexto em que os projetos estão inseridos, suas características e impactos que a integração contínua pode trazer nos mesmos. Em seguida, é analisado o estado da arte dos processos de integração contínua, visando buscar a melhor alternativa possível dentro das restrições do laboratório. Em seguida, é implementado e testado o processo de Integração e Deployment Contínuos em um dos projetos do laboratório, a fim de avaliar a viabilidade e aplicabilidade do processo implantado. Por fim, é criado um guia, para que os desenvolvedores do laboratório consigam implementar os fluxos de integração contínua de maneira fácil em projetos já existentes, ou em projetos futuros. Uma avaliação inicial, realizada com pesquisadores do GQS, levantou indícios de que os guias criados no contexto do Processo de Integração e Deployment Contínuos estavam satisfatórios, e conseguiram transmitir bem as informações necessárias. E o processo de CI/CD de exemplo que foi desenvolvido e implantado no Projeto escolhido estava sendo executado com sucesso.

Abreviaturas e siglas

- **GQS:** Grupo de qualidade de software;
- **CI:** *Continuous integration*, ou integração contínua;
- **CD:** *Continuous Deployment*, ou publicação contínua;
- **Linting:** Processo automático ou manual de verificação de qualidade do código;
- **Deployment:** Entrega ou publicação de um software;
- **Pipeline:** Sequência ordenada de comandos ou processos executados para uma finalidade em comum;
- **Stakeholder:** Parte interessada em um projeto;
- **Snapshot:** Registro do estado de um sistema, aplicação ou arquivos em determinado ponto no tempo.

1. INTRODUÇÃO

Ao começar a desenvolver um software, tipicamente a arquitetura do mesmo é definida. A arquitetura de software consiste em um conjunto de decisões que um time toma sobre um projeto de software, que impactarão diretamente a maneira como o mesmo deve ser construído e mantido (Terra, Túlio 2010).

A arquitetura definida para o software começa, então, a ser implementada, e testes unitários ou de integração, verificações de qualidade de escrita de código, validações de contratos entre API's, *deploy* (implantação) do projeto, auxiliam um time a manter a integridade do software (Góis et. al. 2007).

No entanto, segundo TERRA (2015), na medida que um software cresce, bem como o time que o desenvolve, é quase que natural que o time acabe se desviando das boas práticas de engenharia de software. Testes vão sendo deixados para trás, padrões de arquitetura param de ser seguidos e problemas passam a ocorrer em ambientes de produção. Isso pode ocorrer por uma série de motivos, como prazos curtos, rotatividade do time, pressão dos stakeholders, etc.

Preservar padrões e boas práticas de arquitetura de um software contribui fortemente para manter a escalabilidade, manutenibilidade, reuso e portabilidade de um software (Terra, Túlio 2010). Mas manter esses padrões definidos pelo time manualmente é uma tarefa difícil. E é aí que entram os fluxos de integração e deployment contínuos.

Segundo a norma IEE 2675 (2021), o conceito de Integração Contínua, ou CI, pode ser definida como: "Técnica que continuamente mescla artefatos, incluindo atualizações de código-fonte de todos os desenvolvedores de uma equipe, em uma linha principal compartilhada para construir e testar o sistema desenvolvido."[1] .

Já *deployment* contínuo, ou CD, é o "Processo automatizado de implantação de alterações na produção, realizando validações a fim de reduzir riscos."[2]. (IEE 2675 (2021))

Ou seja, em outras palavras, os fluxos de CI e CD, são fluxos automatizados para realizar alguma rotina em um projeto. Eles facilitam muito a vida do desenvolvedor ou de um time de desenvolvedores, principalmente quando se trata de um projeto grande. Com um fluxo de CI bem estabelecido, um desenvolvedor consegue definir rotinas automáticas que garantam um bom funcionamento de um projeto, ou até que realizem a entrega do mesmo.

O GQS é um grupo de pesquisas do Departamento de Informática e Estatística da Universidade Federal de Santa Catarina. Os pesquisadores do GQS desenvolvem pesquisas na área de engenharia de software, com a finalidade de identificar e elaborar processos para melhorias na qualidade e produtividade no desenvolvimento de softwares (GRUPO DE QUALIDADE DE SOFTWARE, 2022a). O laboratório também possui programas de ensino na área de computação em escolas.

---

[1] tradução própria, original: "*Technique that continually merges artifacts, including source code updates from all developers on a team, into a shared mainline to build and test the developed system.*

[2] "tradução própria, original: "*Automated process of deploying changes to production by verifying intended features and validations to reduce risk.*"

No contexto do GQS é realizada a iniciativa Computação na Escola, que visa levar o ensino de baixo custo sobre computação para as escolas no ensino de níveis fundamental e médio, por meio do planejamento, desenvolvimento, e aplicação de unidades instrucionais (GRUPO DE QUALIDADE DE SOFTWARE, 2022b).

As unidades instrucionais da Computação na Escola são projetadas para levar o ensino sobre a computação para as escolas de maneira interdisciplinar, interligando a computação com outras matérias como Geografia e Linguagens, utilizando ferramentas didáticas e de fácil entendimento para alunos no nível de ensino fundamental ou médio (GRUPO DE QUALIDADE DE SOFTWARE, 2022b).

Assim, no GQS, têm sido desenvolvidos diversos projetos e sistemas para esses fins. Dentre os principais estão os projetos:

- **App Inventor:** Uma ferramenta online de código aberto, originalmente desenvolvida pelo MIT (Massachusetts Institute of Technology, 2022) para a criação de aplicativos mobile gratuitamente, através de uma interface didática e amigável ao usuário. A ferramenta possibilita ao usuário a construção de interfaces de aplicativos arrastando e soltando componentes de front-end, a fim de construir as telas do aplicativo.
- **ScketchToAIA:** Uma ferramenta online que possibilita ao usuário transformar seus *wireframes* em arquivos na extensão **aia**, que podem ser carregados no App Inventor para posteriormente criar um aplicativo funcional.
- **CodeMaster:** Uma ferramenta online que avalia projetos feitos com o App Inventor. O usuário envia um arquivo correspondente ao projeto criado, que pode estar nos formatos ***.xml***, ***.aia***, e ***.jpg***, e o sistema avalia a complexidade do aplicativo, dando uma nota para o mesmo no final.

Os projetos implementados no GQS são gerenciados na plataforma GitLab (GITLAB B. V, 2022a), que é hospedada em um servidor da UFSC. A plataforma GitLab possui um grande suporte para fluxos de integração contínua e deployment contínuo (CI/CD), possuindo uma documentação extensa que facilita bastante para o desenvolvedor implementar essa funcionalidade em novos projetos ou projetos já existentes.

No entanto, apesar dos projetos desenvolvidos no GQS possuírem alto impacto na sociedade, e a plataforma em que os projetos se encontram possua suporte para CI/CD, a grande maioria deles não possui fluxos de CI/CD. As entregas são feitas manualmente, bem como as verificações de testes, sem falar que alguns dos projetos são desenvolvidos por mais de uma pessoa, e ter um fluxo de *linting* nestes casos seria interessante para manter um estilo e qualidade de escrita de código constante.

Espera-se que o desenvolvimento, implantação e documentação de um processo de integração e Deployment contínuos no grupo de pesquisas GQS, possibilite um fluxo de entrega e qualidade constante e de fácil uso.

1.1 OBJETIVOS

1.1.1 OBJETIVO PRINCIPAL

O objetivo principal do presente projeto consiste em desenvolver, documentar e implantar um processo de Integração e Deployment Contínuos no grupo de pesquisas GQS. O processo será desenvolvido a partir da análise das características dos projetos de software desenvolvidos no GQS, da infraestrutura disponível e das suas limitações.

1.1.2 OBJETIVOS ESPECÍFICOS

- Analisar a literatura e o estado da arte em relação a integração e deployment contínuos;
- Analisar os sistemas desenvolvidos e a infraestrutura do GQS;
- Desenvolver e implantar o processo de CI/CD;
- Documentar o processo de CI/CD;
- Avaliar o processo de CI/CD;

1.2 ABORDAGEM METODOLÓGICA

Para alcançar os objetivos, este trabalho realiza uma pesquisa aplicada e para a sua implementação serão realizadas as seguintes etapas metodológicas:

1. **Criação de um processo de CI/CD de exemplo em um projeto do laboratório.**
   Nesta etapa, será escolhido um projeto do laboratório para implementar um fluxo de integração e deployment contínuos de exemplo.
   O pretexto principal desta etapa será escolher um dos projetos mais completos do grupo GQS, a fim de criar uma pipeline de CI/CD o mais completa possível.

2. **Criação de guias de CI/CD**
   Os dados teóricos e práticos levantados com a criação do fluxo de CI/CD de exemplo serão utilizados como material para a criação de guias que serão implementados no grupo GQS para que outros pesquisadores consigam implementar pipelines de CI/CD no futuro.

3. **Avaliação do processo de CI/CD**
   Nesta etapa será aplicada uma pesquisa qualitativa, com a ajuda de outros pesquisadores do laboratório, para avaliar se o processo de CI/CD criado cumpre os objetivos previstos.

4. **Avaliação dos guias**
   Nesta etapa será aplicada uma pesquisa quantitativa em forma de formulário, para avaliar se os guias gerados ao longo do projeto cumprem seus objetivos.

2. FUNDAMENTAÇÃO TEÓRICA

Neste capítulo serão introduzidos os conceitos gerais necessários para o entendimento do problema e da solução proposta, bem como as ferramentas e tecnologias que serão utilizadas ou estarão disponíveis para a implementação da solução, e como estes conceitos se relacionam com o contexto do laboratório GQS, e do projeto proposto.

2.1 CONTROLE DE VERSIONAMENTO DE CÓDIGO

EBERMANN (2019) define esta como uma das partes mais importantes do fluxo de integração e deployment contínuo. É uma metodologia que propõe monitorar quaisquer mudanças feitas em um projeto de software, o colocando em um repositório na internet, acessível publicamente ou pelo time que o desenvolve, possibilitando que o código seja baixado por quem tiver acesso ao repositório.

Desta forma, o controle de versionamento permite que desenvolvedores façam mudanças no código de um projeto de software localmente, e consigam mesclar suas mudanças no código-fonte do projeto sem perder o histórico dos estados anteriores.

No contexto do GQS, o sistema de controle de versionamento de código utilizado é o Git, e a plataforma onde se encontram os repositórios dos projetos é o GitLab.

Existem alguns conceitos chave dentro dessa metodologia que serão explorados neste projeto:

- **Commit:** Representa um conjunto de mudanças em um projeto de software em respeito a um estado anterior (Barik et. al. 2005). Os commits são feitos pelos desenvolvedores, neles existe um registro das mudanças feitas, do autor das mudanças, e data em que as mudanças foram feitas.
- **Branch:** Segundo Schiestl (2020), serve como uma *ramificação* do código do projeto, em outras palavras, é uma cópia do código principal do projeto em que desenvolvedores podem realizar *commits* para salvar seu trabalho, sem alterar o código principal do projeto. Geralmente o código principal se encontra numa *branch* "pai", comumente chamada de *master*.
- **Merge Request:** Após um desenvolvedor encerrar seu trabalho em uma *branch* separada, ele abre um Merge Request, que nada mais é que uma requisição para que o código implementado seja integrado no código principal do projeto. (GITLAB B. V. 2022)

No contexto do GQS, é utilizado a tecnologia Git, e os repositórios dos projetos, como dito anteriormente, estão na plataforma GitLab. Os pesquisadores dos laboratórios fazem o download do código fonte do projeto, que se encontra na *branch* principal chamada de *master,* e criam uma *branch* a partir dessa *branch* principal, para conseguir enviar *commits* com suas alterações ao repositório sem afetar o código principal do projeto em que está trabalhando.

2.2 INTEGRAÇÃO CONTÍNUA (CI)

Segundo EBERMANN (2019), a integração contínua é uma metodologia de **desenvolvimento** de software que ajuda os desenvolvedores a manter a qualidade do código e elementos do mesmo. Ela propõe mesclar artefatos e componentes do software em uma versão central, em um momento posterior ao deploy, e realizar testes nesta versão, para minimizar conflitos ou falhas no ambiente de produção, além de garantir a qualidade do produto implementado.

Como parte principal dessa metodologia, é realizado testes no componente de código gerado. O código é construído e testado, buscando por falhas de regras de negócio, erros na escrita de código, problemas de segurança, erros de integrações com outros sistemas, entre outras coisas.

Ao implementar um processo de integração contínua em um projeto, é possível realizar integrações de código continuamente no mesmo, sem maiores preocupações com a qualidade e integridade das funcionalidades do produto de software.

Estes processos de integração contínua podem ser feitos de diversas maneiras, alguns times de desenvolvimento utilizam processos totalmente manuais, outros utilizam processos parcialmente ou totalmente automatizados.

Neste projeto, objetivaremos a construção de processos de integração contínua parcialmente ou totalmente automatizados, e os guias de construção destes processos serão criados com base nesta premissa.

2.3 DEPLOYMENT CONTÍNUO (CD)

De acordo com EBERMANN (2019) deployment contínuo é uma metodologia de **entrega** de software, ela serve como uma etapa posterior a integração contínua, propondo uma entrega automática do componente de software gerado no passo anterior, assim que todas as validações necessárias forem cumpridas com sucesso.

Realizar verificações e publicações de entregas continuamente, idealmente após cada novo commit em um projeto de software, é o que chamamos de pipeline do fluxo de CI/CD.

Na figura 1 é apresentado um fluxo genérico que exemplifica a pipeline de CI/CD. Esta pipeline continuamente testa, constrói, e publica artefatos, assim automatizando completamente o processo de entrega do software, da etapa do commit até o produto acessível ao usuário final em um servidor ou na nuvem.

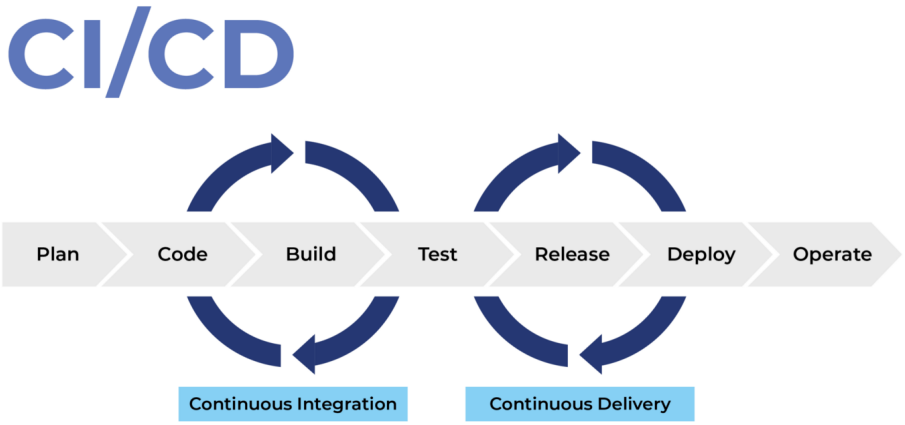


Figura 1: fluxo genérico de CI/CD
(Fonte: https://www.clouddefense.ai/blog/how-to-implement-effective-ci-cd-pipeline)

Ainda de acordo com EBERMANN (2019), este fluxo de entrega publicação de software não se limita apenas ao ambiente de produção, na verdade segundo LE YEOH, J. (2021), usualmente existem três tipos de ambientes na qual o software pode ser publicado, cada com com sua utilidade, e base de dados separadas:

- **Ambiente de integração:** ambiente onde o código é publicado temporariamente, geralmente durante a duração da pipeline de CI/CD após um commit, para testes automáticos.
- **Ambiente de teste (usualmente chamado de *Staging*):** ambiente onde uma versão completa do código é publicada logo antes da publicação para produção, para que quaisquer alterações novas sejam testadas, ou até para demonstrar a algum stakeholder quais novidades entrarão no sistema.
- **Ambiente de produção:** ambiente final do software, onde as alterações testadas estarão disponíveis ao usuário final.

Todo esse fluxo de entrega visa diminuir o máximo possível o número de falhas no ambiente de produção, bem como a qualidade, manutenibilidade e escalabilidade do projeto.

No contexto do GQS, alguns projetos possuem um ambiente de desenvolvimento, numa branch chamada ***dev***. Desenvolvedores criam branches para implementar novas funcionalidades e criam Merge Requests para esta branch. Assim que este Merge Request é aceito, o novo código implementado é integrado a esta branch.

Após essa nova funcionalidade ter sido testada, o código é integrado a branch ***master***, e é publicada uma nova versão do sistema no ambiente de produção.

2.4 DESENVOLVIMENTO ÁGIL

EBERMANN (2019) define o conceito de desenvolvimento ágil como sendo: "Uma abordagem flexível em desenvolvimento de software onde um produto evolui com um diálogo constante entre desenvolvedores e clientes, ao invés de ter uma definição inicial pesada"[3].

O desenvolvimento ágil é um conceito importante no escopo atual do desenvolvimento de software, ele propõe a incrementação gradual e contínua de um software conforme o produto, time, ou até a empresa em que o projeto está inserido, vão evoluindo e se desenvolvendo.

Este conceito pode ser facilmente relacionado com os conceitos de integração contínua e deployment contínuo. Ao implementar o conceito de desenvolvimento ágil em uma equipe que implementa um produto de software, este produto em questão sofrerá alterações continuamente, conforme os requisitos de produto vão evoluindo. Sem mencionar que, possivelmente, o time que desenvolve este produto sofrerá mudanças de recursos humanos ao longo do tempo, fazendo com que o software seja desenvolvido por pessoas diferentes ao longo de seu tempo de vida. Visto isso, implementar um fluxo de integração e deployment contínuo no projeto em questão ajudará a garantir a integridade e conformidade arquitetural do projeto.

Existem uma série de metodologias que implementam o desenvolvimento ágil, como o scrum e o kanban, metodologias essas que são chamadas de metodologias ágeis.

No contexto do GQS, não são necessariamente implementadas metodologias ágeis para o desenvolvimento dos produtos de software do laboratório, porém, o conceito de desenvolvimento ágil pode ser aplicado, ao passo de que os softwares implementados recebem incrementações graduais.

Os projetos do GQS são comumente desenvolvidos ou refatorados por alunos da Universidade Federal de Santa Catarina (UFSC), em sua maior parte, durante o desenvolvimento de trabalhos de conclusão de curso, mestrado ou doutorado.

Isso faz com que haja uma alta rotatividade dos recursos humanos disponíveis para o desenvolvimento dos projetos, sem mencionar que, a cada novo projeto, existe um novo requisito que será implementado no contexto do GQS, seja refatorando um projeto já existente, adicionando uma nova funcionalidade, ou até criando um projeto novo.

Visto isso, o contexto do desenvolvimento no laboratório GQS pode ser facilmente relacionado com a realidade de um software que utiliza metodologias ágeis. Por isso, a implementação de um fluxo de integração contínua e/ou deployment contínuo poderá vir a ser benéfico para a qualidade das entregas e dos desenvolvimentos dos softwares do laboratório.

---

[3] tradução própria, original: "A flexible approach in software development where a product evolves through continuous dialogue between developers and clients or customers rather than a heavyweight initial specification"

2.5 GITLAB

O GitLab (GITLAB B.V., 2022a) é a plataforma onde se hospedam os repositórios dos projetos do GQS. Essa ferramenta é utilizada para hospedar repositórios e realizar o controle de versionamento dos mesmos. Ela utiliza a ferramenta Git para controle e versionamento do código dos repositórios.

A plataforma também possui uma série de soluções para desenvolvedores, como por exemplo:

- Suporte para wiki's e documentações;
- Suporte para integração contínua;
- Suporte para deployment contínuo.

A ferramenta pode ser acessada utilizando a plataforma online do GitLab, onde os repositórios dos projetos serão guardados, administrados e disponibilizados através do host do GitLab, ou também pode ser baixada em uma máquina separada, onde quem será responsável por administrar, monitorar e disponibilizar a plataforma será o administrador da máquina e/ou do projeto.

No contexto do GQS, o GitLab está instalado em uma máquina da SeTIC, e é disponibilizado pelo endereço https://codigos.ufsc.br, onde alunos e/ou pesquisadores cadastrados podem se autenticar e acessar repositórios de projetos a qual eles possuem acesso.

Visto isso, o GitLab será a plataforma principal onde o projeto proposto será desenvolvido, visto que ela possui todo o suporte necessário para a implementação dos objetivos do projeto, e é onde os softwares estão hospedados.

2.6 GITLAB CI

O GitlabCI é a ferramenta de integração e deployment contínuo própria do GitLab, com ela é possível implementar pipelines de CI/CD.

2.6.1 PIPELINES

De acordo com a documentação do GitLab CI (GITLAB B.V. 2022), essas pipelines são implementadas em um arquivo com a extensão .yml, que fica dentro de cada projeto, chamado gitlab-ci.yml.

Essas pipelines são compostas de:

- **Jobs:** Que define o que deve ser feito, como um job para realizar um deploy ou rodar os testes, por exemplo.
- **Stages (ou estágio):** É composto por jobs, e é usado para definir em que ordem os jobs devem rodar.

As pipelines são compostas de vários *Stages*, que são compostos por uma série de *jobs*. Os *stages* são executados na ordem definida no arquivo gitlab-ci.yml, e seus *jobs* são executados em paralelo. Quando todos os *jobs* de um *stage* são executados com sucesso, a pipeline passa para o próximo *stage*.

Quando uma pipeline quebra em algum *stage* ou *job*, o processo encerra, e o

GitLab manda automaticamente um email para o desenvolvedor que iniciou a pipeline indicando o que ocorreu, para que ele possa identificar o problema.

No arquivo mostrado na figura 2, pode-se ver que temos 4 *jobs* (*build, unit_tests, integration_tests, deploy*) e 3 *stages* (*build, test, deploy*).

```yaml
image: alpine

build:
  stage: build
  script:
    - echo "building..."
    - echo "built!"

unit_tests:
  stage: test
  script:
    - echo "running unit tests..."
    - echo "tested!"

integration_tests:
  stage: test
  script:
    - echo "running integration tests..."
    - echo "tested!"

deploy:
  stage: deploy
  script:
    - echo "deploying app..."
    - echo "deployed!"

```

Figura 2: Exemplo de um arquivo gitlab-ci.yml.
(Fonte: elaborado pelo autor)

Na figura 3 pode-se ver a pipeline da figura 2 em execução. Como informado, primeiramente é executado o *step* de ***build***, então é executado o *step* de *test*, onde os *jobs* de ***unit_tests*** e ***integration_tests*** são executados em paralelo, e por fim é executado o *step* de ***deploy***.

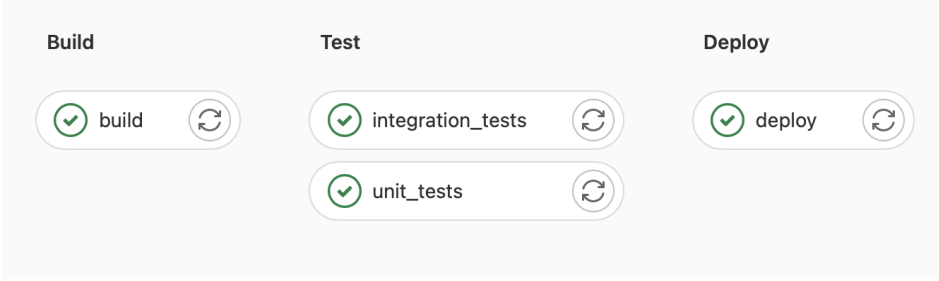


Figura 3: Execução de um processo de CI.
(Fonte: elaborado pelo autor)

Estas pipelines são lidas quando o processo de CI inicia, e o runner do projeto é encarregado de interpretar e executar os comandos da pipeline.

2.6.2 RUNNERS

Os ***runners*** são aplicações que rodam dentro de máquinas externas à máquina onde está rodando o GitLab, e são configurados no projeto para executar os scripts definidos no arquivo ***gitlab-ci.yml*** do mesmo.

Segundo a documentação oficial do GitLab (GITLAB B.V. 2022), é instalado a aplicação do runner na máquina desejada, e através desta aplicação é possível registrar o runner em um projeto do GitLab.

As duas máquinas, do GitLab e do runner, não precisam estar na mesma rede, mas a máquina do runner deve estar disponível quando o projeto tentar executar um processo de CI.

O processo de CI utilizará os recursos da máquina onde o runner está para executar a pipeline. Por isso, se uma pipeline utiliza algum recurso, como Java ou Docker por exemplo, a máquina configurada precisa conter esse recurso, se ele não puder ser configurado durante a própria pipeline.

O runner é configurado utilizando a url onde o GitLab se encontra, e um token de registro, que são encontrados na página Na aba ***settings > CI/CD > runners*** do projeto.

Na figura 4 é possível ver a interface onde estes dados podem ser encontrados, incluindo o token de registro do Runner.

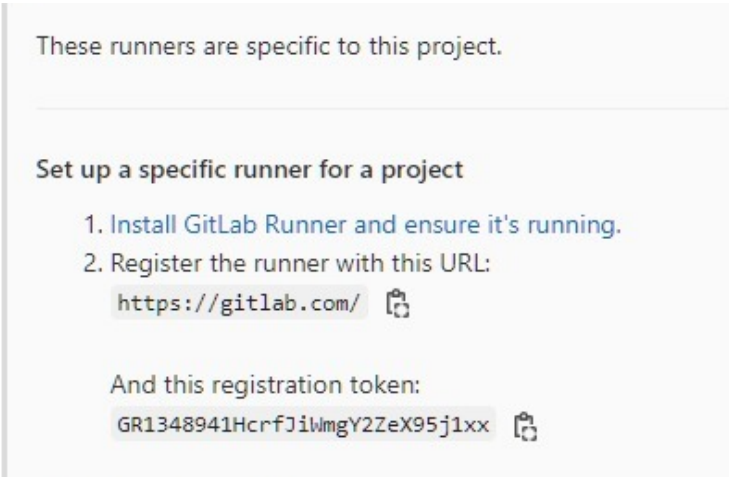


Figura 4: Exemplo de dados para configurar um runner.
(Fonte: elaborado pelo autor)

O runner tem os seguintes atributos:

- **descrição:** um texto opcional que identifica o runner, como informações sobre a máquina em que ele se encontra, por exemplo, para fins de documentação.

- **tags:** Um projeto pode ter mais de um runner. Neste caso, é preciso ter uma maneira de distinguir, na pipeline, qual runner deve ser utilizado em um job, e para isso existem as tags. As tags são textos opcionais que identificam o runner.

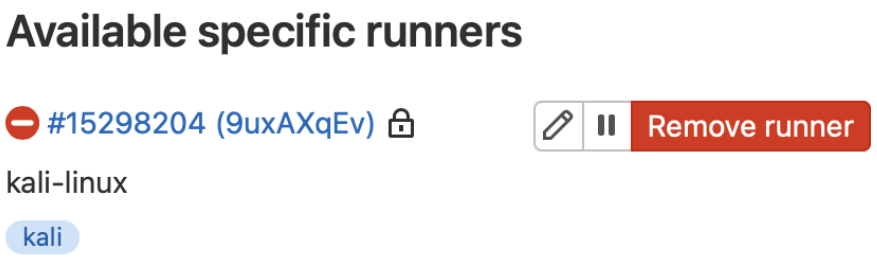


Figura 5: Exemplo de runner com tag.
(Fonte: elaborado pelo autor)

Por exemplo este runner com a tag ***kali*** mostrado na figura 5, se fosse necessário usá-lo apenas no job ***deploy*** da figura 2, poderíamos realizar isso com o seguinte código:

```
21  deploy:
22    tags:
23      - "kali"
24    stage: deploy
25    script:
26      - echo "deploying app..."
27      - echo "deployed!"
```

Figura 6: Exemplo de uso de tags na pipeline.
(Fonte: elaborado pelo autor)

Desta forma, o job de ***deploy*** utilizará apenas o runner especificado.

- **executor:** O executor determina qual ambiente o runner utilizará para rodar a pipeline. Por exemplo, se o desenvolvedor quiser escrever o script utilizando a linguagem do shell, ele pode usar o executor **shell**, se ele preferir escrever utilizando ssh, ele utilizará o executor **ssh.**

No contexto do GQS, o projeto Sketch2AIA já possui um runner configurado, este runner está instalado em uma máquina na SeTIC. Por estar na mesma rede da UFSC, este runner consegue se conectar à máquina onde os projetos se encontram

e realizar os ***jobs*** de deploy automaticamente, bem como utilizar os recursos daquela máquina, como o tomcat, por exemplo.

2.7 DOCKER

Segundo a documentação oficial (DOCKER INC 2021), o Docker é uma ferramenta aberta para agilização do desenvolvimento de produtos de software. Com o Docker é possível separar os recursos de infraestrutura de uma aplicação (como bancos de dados, dependências de software como Java e Python, entre outras coisas), da aplicação em si, de uma forma reutilizável entre os desenvolvedores de um projeto.

O Docker realiza isso utilizando contêineres. Estes contêineres são ambientes isolados que rodam todas as dependências de uma aplicação, possuindo seus recursos de infraestrutura, dependências de software, e até a aplicação em si.

Desta maneira desenvolvedores conseguem configurar contêineres do Docker em um projeto de software que possuam tudo o que é necessário para rodar a aplicação, e compartilhar estes contêineres entre os desenvolvedores do projeto. E então os desenvolvedores apenas precisam rodá-los utilizando o Docker, sem precisar baixar ou subir nenhum recurso de infraestrutura ou software em suas máquinas, assim facilitando o processo de configuração e desenvolvimento de aplicações.

O uso principal desta tecnologia é tornar possível o compartilhamento fácil dos ambientes de desenvolvimento de uma aplicação entre desenvolvedores, para que membros de uma organização não precisem se preocupar com as configurações das máquinas antes de começar a desenvolver.

Contudo, o uso do Docker vai além. Quando é realizada a entrega de uma aplicação em ambientes de produção, é comum os desenvolvedores se preocuparem com quais configurações estão presentes nas máquinas de produção, quais recursos de software existem, qual sistema operacional, etc. O Docker pode ser usado para facilitar este processo, possibilitando assim subir uma aplicação ou um ambiente de execução apenas utilizando o Docker. Desta maneira, o time apenas precisa se preocupar se as máquinas de produção possuem o Docker, o que pode ajudar muito em projetos com muitas tecnologias diferentes.

O conceito de contêiner pode ser facilmente comparado com uma virtualização de uma máquina. Os dois executam instâncias de sistemas operacionais dentro de hospedeiros (que são as máquinas dos desenvolvedores ou de produção), com recursos de software e infraestrutura. Porém a conteinerização é uma versão mais leve e dedicada, ela apenas executa uma instância contendo os recursos estritamente necessários para a execução de uma aplicação.

Os contêineres do Docker são construídos e executados através de imagens. As imagens são templates, ou scripts, para a execução de contêineres do Docker. Os desenvolvedores podem usar imagens já prontas disponíveis em repositórios públicos do Docker em casos de uso mais simples, ou podem criar as suas próprias imagens.

Quando é necessário criar uma imagem Docker personalizada para o projeto, isto deve ser feito através de um arquivo ***Dockerfile***. Estes arquivos são basicamente scripts que indicam ao Docker o que deve ser feito para construir uma imagem do Docker, para que então ela seja executada em formato de contêiner.

Para exemplificar, supondo um projeto hipotético que roda utilizando Node.js, com uma dependência do software python, seria possível construir uma imagem Docker para ele escrevendo um ***Dockerfile*** neste formato:

```
FROM node:12-alpine

RUN apk add --no-cache python2 g++ make

WORKDIR /app
COPY . .
RUN yarn install --production
CMD ["node", "src/index.js"]
EXPOSE 3000
```

(Fonte: https://docs.docker.com/get-started/02_our_app/)

O arquivo então indica ao Docker que, para construir a imagem Docker, é preciso:

- Utilizar como base a imagem Docker já pronta do node versão **12-alpine**;
- Rodar um comando para adicionar a dependência do software python;
- Instalar e rodar a aplicação;
- Expor a aplicação e seus componentes executados na porta 3000 do contêiner.

As imagens personalizadas normalmente utilizam como base imagens já prontas, que estão disponíveis em repositórios na web (sejam públicos ou privados), como o Docker hub, e então incrementam estas imagens com outros recursos de infra ou de software necessários para a execução de uma aplicação. Estas imagens são carregadas utilizando o comando **FROM** do dockerfile.

Ainda utilizando o exemplo em questão, supondo que seja necessário construir uma imagem Docker de nome "exemplo-docker" usando este dockerfile, seria possível então construí-la utilizando a aplicação Docker, com o comando:

```
$ docker build -t exemplo-docker .
```

Após isso, a imagem Docker seria acessível na máquina em questão através do Docker. Sendo assim, supondo que seja necessário rodar esta imagem Docker na porta **8080** da máquina hospedeira, seria possível executá-la como um contêiner através do comando:

```
$ docker run -p 8080:3000 exemplo-docker
```

Lembrando que foi indicado, no Dockerfile, que a aplicação deveria rodar na porta 3000 do contêiner e, como existe a necessidade de rodar a aplicação na porta 8080 da máquina, foi utilizado o parâmetro -p para indicar ao Docker que a porta 3000 do contêiner deve ser mapeada à porta 8080 da máquina em que o contêiner está sendo rodado.

Como dito anteriormente, existe a possibilidade também de executar uma imagem Docker já pronta. Nestes casos, isso pode ser realizado apenas com o comando ***docker run***.

Supondo que exista a necessidade de rodar apenas um recurso de infra, um banco de dados ***MySQL***, na versão 8.0.29. Seria possível realizar isso buscando a sua imagem Docker em um repositório, como no caso, o Docker hub, e então usar a imagem Docker como parâmetro do comando ***docker run:***

```
$ docker run mysql:8.0.29
```

Como já informado anteriormente, e como é possível ver na **Figura 2**, as pipelines do gitlab-ci podem ser executadas utilizando imagens do Docker, possibilitando que os comandos sejam executados em ambientes diferentes conforme a necessidade do desenvolvedor. Por este motivo, o Docker será amplamente utilizado neste projeto para implementar os fluxos de integração contínua.

3. PROPOSTA DE SOLUÇÃO

Neste capítulo é apresentada a proposta de solução a qual esse trabalho objetiva.

A proposta principal deste projeto consiste no desenvolvimento e documentação de um processo de CI/CD para os projetos do grupo de pesquisas GQS. O processo é documentado no formato de um guia, para que os pesquisadores do grupo consigam implementar estes fluxos nos projetos novos ou já existentes.

Para a confecção deste guia, é realizada a implementação do fluxo de CI/CD em um projeto de exemplo, para que sirva como base teórica e prática da documentação. Visto isso, este capítulo inicia com definição da estrutura do guia de CI/CD que será desenvolvido e em seguida é apresentada, como exemplo, a implementação de um processo de CI/CD em um dos projetos já existentes do laboratório GQS.

Dentre os projetos que estão inseridos dentro do contexto do GQS, este capítulo apresenta como exemplo a definição de um processo de CI/CD no projeto CodeMaster (GRUPO DE QUALIDADE DE SOFTWARE, 2021). Esse projeto foi escolhido por ser um dos mais completos desenvolvidos no grupo em termos de tecnologias e complexidade, possuindo vários subprojetos implementados em tecnologias diferentes. Além disso, atualmente esse é o projeto de software mais ativo desenvolvido no grupo, envolvendo pesquisadores de graduação, mestrado e doutorado.

3.1 DEFINIÇÃO DO GUIA DE CI/CD

O processo definido neste TCC é documentado na forma de um guia contendo um passo a passo de como implementar um fluxo de CI/CD em projetos do laboratório. Este guia é confeccionado utilizando a mesma plataforma onde o projeto de exemplo está hospedado, o GitLab, utilizando a ferramenta ***wiki*** (GITLAB B.V, 2022g) da plataforma para a confecção dos textos.

O guia de CI/CD será separado em uma série de sessões, que são definidas neste capítulo.

3.1.1 SEÇÃO 1: COMO FUNCIONA O GITLAB-CI

Nesta seção será ser introduzida a parte teórica de como funciona o processo de CI/CD na plataforma GitLab (gitlab-ci), contendo, no mínimo, os seguintes tópicos:

- **Como funciona a linguagem do gitlab-ci.yml:** deverá explicar os principais componentes da linguagem utilizada nos scripts do gitlab-ci, e como eles se comportam em um fluxo em execução;

- **O que são runners, e quais são seus componentes:** deverá introduzir o conceito de runners do GitLab, para que servem, e quais são seus

componentes (como executores, sistema operacional, tags, etc);

- **O que são executores:** deverá explicar de maneira detalhada o que são os executores dos runners do GitLab, e qual a importância deles para execução dos fluxos de CI/CD;

- **O que é Docker, e como ele se relaciona com o fluxo de CI/CD:** deverá introduzir a ferramenta Docker, como ela funciona, e qual a relação dela com os executores e os fluxos de CI/CD do gitlab-ci.
  Deverá também possuir exemplos de Dockerfiles explicando as funcionalidades da linguagem Docker.

Esta parte introdutória será importante para desenvolvedores iniciantes entenderem melhor sobre as funcionalidades e ferramentas disponíveis, para então poderem seguir o guia de configuração de CI/CD com um melhor entendimento do que estão fazendo.

3.1.2 SEÇÃO 2: CONFIGURANDO CI/CD EM UM PROJETO

Esta seção apresentará o passo a passo necessário para configurar um projeto no GitLab para rodar um fluxo de CI/CD. Indicando:

- **Quais as configurações que devem ser feitas no projeto para possibilitar o uso do gitlab-ci:** antes de configurar runners do GitLab ou escrever as pipelines, existem configurações que devem ser feitas no projeto para possibilitar o uso do gitlab-ci. Este tópico deverá descrever com detalhes quais são essas configurações, contendo exemplos visuais e práticos;

- **Como configurar um runner no projeto:** após as configurações iniciais serem feitas no projeto, é necessário configurar um runner para o projeto. Este tópico deverá conter um passo a passo detalhado das ações necessárias para configurar um runner em um projeto do GitLab, contendo imagens exemplificando cada passo;

- **Como testar a configuração e verificar se está funcionando:** com um runner configurado, é conveniente escrever um fluxo de pipeline básico para testar a configuração feita. Este tópico deverá ter um exemplo de arquivo *gitlab-ci.yml* básico, contendo os passos básicos de execução do fluxo de CI, bem como um passo a passo de como implementar e rodar a pipeline.

3.1.3 SEÇÃO 3: CONFIGURANDO UMA IMAGEM PARA O PROJETO

Configurar uma imagem Docker contendo as dependências de software necessárias para rodar o projeto é um passo importante na configuração da pipeline de CI/CD. Visto isso, a sessão 3 conterá o passo a passo de como configurar um imagem Docker base para utilizar nas pipelines de CI/CD do projeto, indicando:

- **Como escrever uma imagem Docker:** este tópico deverá indicar como utilizar a linguagem Docker, que foi introduzida na seção 1, para criar imagens customizadas dos projetos contendo os requisitos de software necessários para rodar as pipelines do projeto alvo.

- **Como realizar o upload da imagem Docker criada para o Docker hub:** com a imagem Docker pronta, é preciso realizar o upload dela para o Docker hub. Este tópico deverá ter um passo a passo indicando como realizar isso.

3.2 DEFININDO CI/CD PARA O PROJETO CODEMASTER

Com o objetivo de implementar uma pipeline de CI/CD em um projeto de exemplo, foi necessário selecionar um projeto do laboratório e, como informado, o projeto escolhido foi o CodeMaster. Este projeto está hospedado no **GitLab**, e o fluxo de integração contínua será implementado utilizando o ***gitlab-ci***, que utilizará uma imagem Docker para rodar as pipelines.

Para rodar a pipeline que será desenvolvida, primeiramente foi necessário entender melhor os componentes do projeto, e suas dependências de software, para então poder implementar uma imagem Docker base, que será utilizada para a execução da pipeline.

3.2.1 ENTENDENDO O CODEMASTER

Na primeira parte do desenvolvimento da imagem Docker base, foi necessário entender mais a fundo como o projeto CodeMaster funciona, suas tecnologias e o contexto do seu desenvolvimento.

Como já dito anteriormente, o CodeMaster é uma ferramenta online que avalia projetos feitos com o App Inventor. O usuário envia um arquivo correspondente ao projeto criado, que pode estar nos formatos ***.xml***, ***.aia***, ***.jpg*** ou ***.tm*** e o sistema avalia a o aplicativo, dando uma nota para diversos conceitos do mesmo no final. O CodeMaster é composto de diversos componentes, conforme mostra a Figura 7.

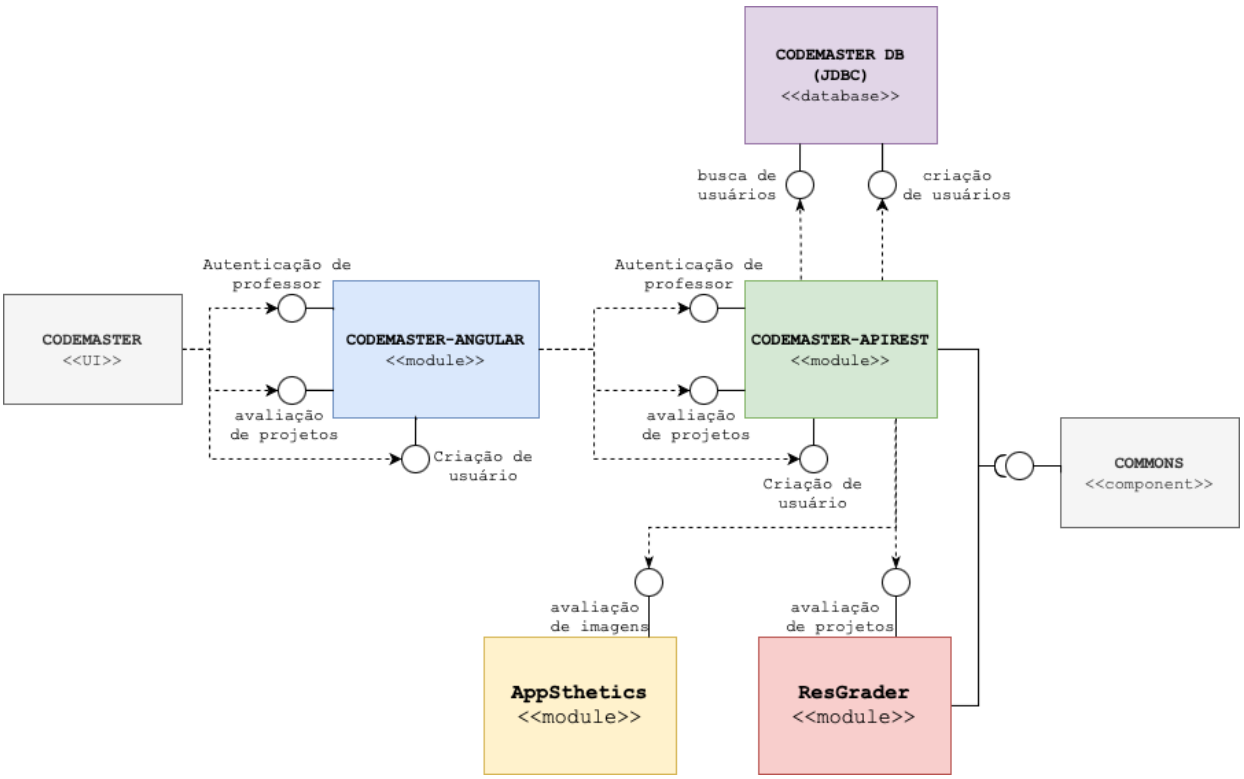


Figura 7: Componentes do CodeMaster.
(Fonte: elaborado pelo autor)

O CodeMaster, em sua versão mais atual, é composto por 5 subprojetos:

- **Commons:** Um projeto escrito em **Java** com bibliotecas e soluções gerais que são utilizadas pelo **codemaster-apirest** e o **RestGrader.**
- **codemaster-apirest:** Um projeto escrito em **Java** utilizando o framework **Spring**, que implementa uma API Rest para lidar com as requisições do **codemaster-angular**, e chama os projetos **RestGrader** e **Appsthetics** para processar as notas dos projetos.
  O projeto também se conecta a um banco **SQL** para salvar e processar logins de usuários.
- **RestGrader:** Um projeto escrito em **Java** que implementa uma API Rest para computar as notas dos projetos enviados pelo **codemaster-apirest** de acordo com parâmetros variados.
- **Appsthetics:** Um projeto escrito em **Python** que implementa uma API Rest com soluções de machine learning para computar as notas dos projetos enviados pelo **codemaster-apirest** que forem enviados no formato ***.jpg*** ou ***.tm***. O projeto utiliza estas soluções de machine learning para computar as imagens e dar uma nota com base na "estética" do projeto.
- **codemaster-angular:** É o frontend do projeto, escrito da tecnologia **angular**, que chama o projeto **codemaster-apirest** através de interfaces HTTP para processar login de usuários, computar as notas, dentre outros.

3.2.2 DEPENDÊNCIAS DE SOFTWARE

Tendo os componentes e funcionamento do projeto mapeados, é necessário identificar e analisar quais as dependências de tecnologias que o projeto possui, sendo elas:

- **Java jdk 1.8:** Utilizado pelos projetos escritos na linguagem Java;
- **maven (versão mais atual):** Utilizado para compilar os projetos escritos em Java;
- **tomcat 8:** Utilizado para realizar o ***deploy***, seja em produção ou local, dos projetos Java;
- **opencv 3.4.3:** Utilizado pelo projeto **RestGrader** para computar alguns parâmetros de notas**;**
- **Python 3:** Utilizado pelo **appsthetics**;
- **pip 3:** Utilizado para gerenciar os pacotes do **appsthetics;**
- **NodeJS (versão mais atual):** Utilizado para rodar o **codemaster-angular;**
- **npm:** Utilizado para gerenciar os pacotes do projeto **codemaster-angular;**

3.2.3 CRIAÇÃO DE UMA IMAGEM DOCKER PARA O PROJETO

Utilizando as informações coletadas, é definida a imagem Docker base que será utilizada para rodar as pipelines do projeto CodeMaster, bastando integrar as dependências de software em um arquivo Dockerfile.

No trecho de código a seguir, é apresentado o esboço de arquivo ***Dockerfile*** que será gerado para a criação da imagem Docker, com os passos necessários numerados para facilitar a explicação posterior:

```
#1
FROM tomcat:8-jdk8
#2
RUN apt-get update && apt-get install -y \
   maven \
   python3-dev \
   python3-pip \
   nodejs \
   npm \
   git-all \
   ant \
   cmake
#3
RUN npm install -g n && n stable
#4
RUN cd ~ && git clone https://github.com/opencv/opencv.git && \
 cd opencv && \
 git checkout 3.4.3 && \
 mkdir build && \
 cd build && \
 export ANT_HOME=/usr/share/ant && \
 cmake .. -DBUILD_SHARED_LIBS=OFF && \
 make -j8 && \
 mvn install:install-file -Dfile=bin/opencv-343.jar -DgroupId=org.openpnp -DartifactId=opencv
-Dversion=343 -Dpackaging=jar
```

- **Passo 1:** Foi utilizado o comando **FROM** para utilizar como base uma imagem Docker já pronta do tomcat, que já contém especificamente o jdk necessário (jdk 8). Esta imagem Docker se encontra no Docker hub (https://hub.docker.com/tomcat/), e utiliza o sistema operacional linux;
- **Passo 2:** Foi utilizado o comando **RUN** para baixar todas as dependências de software restantes (com exceção do opencv) com o apt-get.
  As dependências ***git-all, ant*** e ***cmake*** são dependências do opencv que serão utilizadas posteriormente;
- **Passo 3:** Foi baixado o gerenciador de versões do node ***n*** através do npm, para que seja possível instalar e utilizar sempre a versão mais atualizada do node, sem depender do que foi instalado no passo 2;
- **Passo 4:** O **opencv** não pode ser instalado via apt-get, visto que ele deve ser compilado diretamente utilizando o código fonte do projeto. Por isso, foi utilizado o tutorial de instalação oficial do openCV, disponível na url https://docs.opencv.org/4.x/d9/d52/tutorial_java_dev_intro.html, para instalar e compilar o **opencv;**
  Ao final, com os executáveis do **opencv** instalados, foi adicionado o pacote ao maven local para uso posterior dos projetos Java.

3.2.4 DEPLOY DA IMAGEM PARA O DOCKER HUB

Por fim, com o Dockerfile pronto para geração da imagem Docker base para o projeto gerada, é realizado o upload da imagem Docker construída para o site Docker hub (DOCKER INC, 2021b).

O Docker Hub é um site que visa guardar e disponibilizar imagens do Docker, de forma que estas imagens possam ser usadas como base em outras imagens Docker, ou executadas para gerar contêineres.

Para realizar um upload da imagem Docker base do CodeMaster para o Docker hub, primeiramente é necessária a criação de uma conta para o laboratório GQS, para administração de quaisquer imagens relacionadas ao laboratório. Então, com a conta criada, é preciso criar um repositório relacionado a imagem Docker que será enviada na plataforma do Docker hub (Figura 8 e 9).

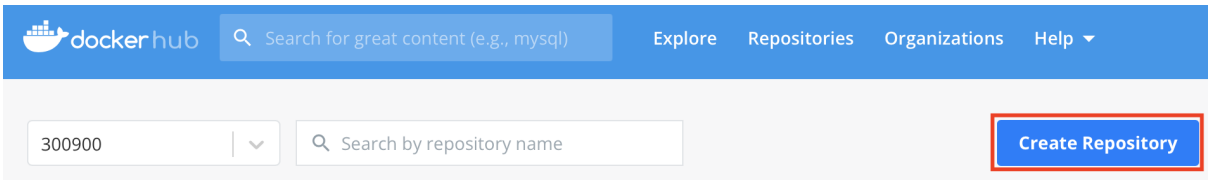


Figura 8: Exemplo de criação de repositório no Docker Hub (parte 1)
(Fonte: elaborado pelo autor)

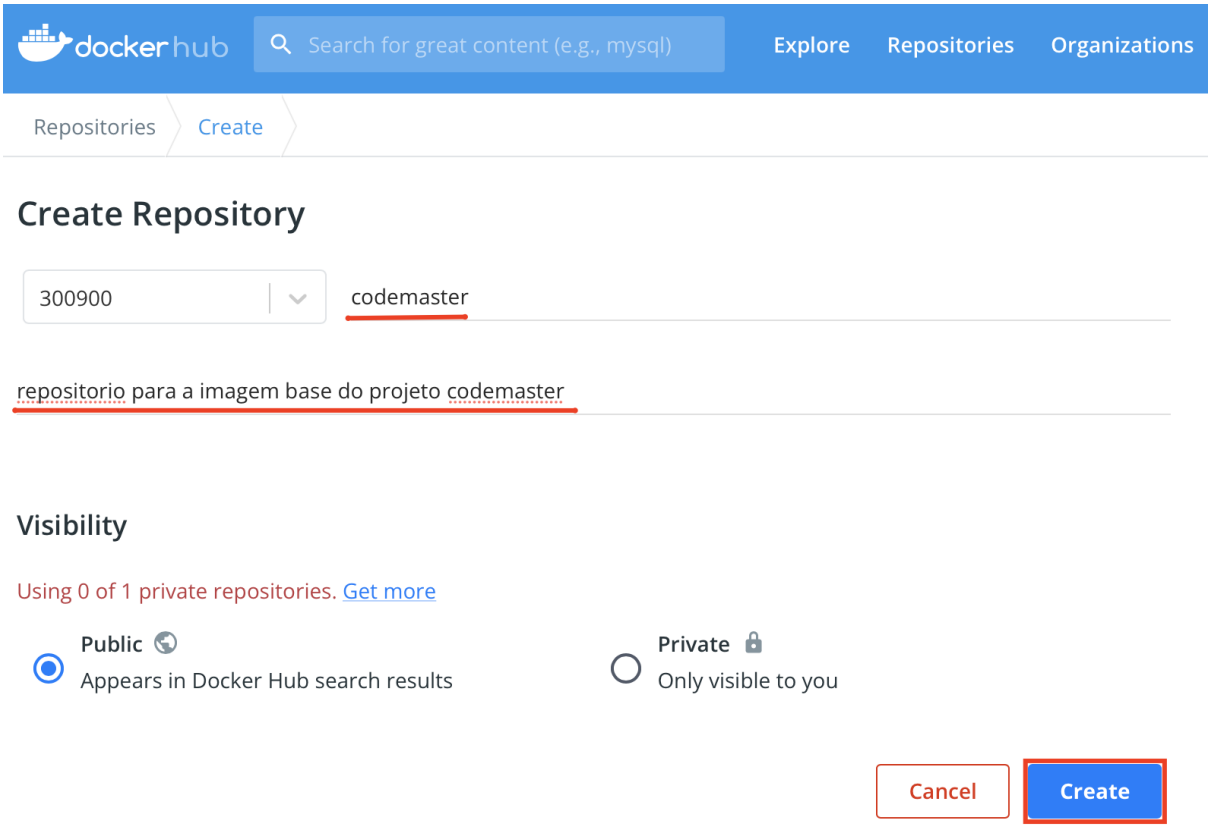


Figura 9: Exemplo de criação de repositório no Docker Hub (parte 2)
(Fonte: elaborado pelo autor)

Com o repositório criado, é possível realizar o upload da imagem Docker para o docker-hub via linha de comando.

Por exemplo, para realizar o upload da imagem Docker base do CodeMaster gerada, que recebeu o nome ***codemaster-base***, no repositório criado nas imagens de exemplo (***codemaster***), são utilizados os seguintes comandos:

1. Realizar login na conta criada para o laboratório:

```
$ docker login --user={usuário GQS} --password={senha GQS}
```

2. Criar uma tag para a imagem Docker na máquina local, relacionando com o repositório:

```
$ docker tag codemaster-base codemaster:v1
```

3. Realizar o push da imagem Docker:

```
$ docker push codemaster:v1
```

Desta maneira, está disponível a imagem Docker já construída em um repositório armazenado na nuvem, sendo possível utilizá-la nas pipelines de integração contínua facilmente.

3.2.4 DEFININDO AS PIPELINES DO CODEMASTER

Com a imagem Docker base carregada no Docker Hub, se pode criar uma pipeline de CI/CD para o projeto CodeMaster, utilizando o GitlabCI.

Para isso, precisaremos seguir os seguintes passos:

1. **Configurar o CodeMaster para utilizar CI/CD:** Realizar as configurações necessárias, na página do projeto no GitLab, para habilitar o CI/CD no projeto;
2. **Configurar um runner para o projeto:** Após a configuração inicial, é preciso configurar um runner para rodar as pipelines. Esta etapa é alinhada com os administradores do laboratório GQS, para melhor entender quais recursos estão disponíveis para isso.
3. **Implementar as pipelines:** Por fim, é necessário implementar as pipelines de CI/CD, criando um arquivo ***gitlab-ci.yml*** na raiz do projeto CodeMaster.
   Esta pipeline deverá ter, no mínimo, os seguintes passos:
   a. Carregamento do código fonte do CodeMaster do GitLab para a máquina de desenvolvimento;
   b. Construção de cada subprojeto e artefato que compõe o CodeMaster;
   c. *Deploy* do projeto para o ambiente de desenvolvimento.

Toda a construção desta pipeline foi utilizada como material para a construção do guia de CI/CD, e também para a provação da viabilidade e valor da implementação destas pipelines em outros projetos do laboratório. A implementação dessa pipeline é apresentada no capítulo seguinte.

4. IMPLEMENTAÇÃO DO PROJETO

Neste capítulo será apresentada como foi feita a implementação do projeto. Explicando como foram colocadas em prática todas as teorias e conceitos que foram apresentados até agora.

Como dito anteriormente, o primeiro passo do projeto consiste na implantação de um processo de CI/CD em um dos projetos do grupo de pesquisas. Visto isso, este capítulo inicia com a apresentação de todo o processo para a criação e implantação de um fluxo de CI/CD no projeto CodeMaster do GQS, indicando com detalhes quais foram as etapas necessárias para a preparação, testes e implantação do mesmo.

Após ter o fluxo de integração contínua criado, todas as informações necessárias para a criação do guia foram coletadas, por isso, ao final deste capítulo será indicado como estes conhecimentos recolhidos foram utilizados para a confecção de um guia de configuração de pipelines de integração e deployment contínuos para todos os desenvolvedores do laboratório GQS.

4.1 CRIAÇÃO DE UM FLUXO DE INTEGRAÇÃO CONTÍNUA

Como dito anteriormente, para recolher as informações práticas necessárias para criar um guia para criação de pipelines de CI/CD de qualidade, é necessário implementar, do começo ao fim, um fluxo de integração contínua em um projeto do GQS como exemplo.

O projeto escolhido foi o CodeMaster, por ser um dos projetos mais complexos do laboratório, com maior número de componentes, linguagens diferentes que o compõem e de usuários, conforme já justificado.

4.1.1 CRIAÇÃO DE UMA IMAGEM BASE PARA O CODEMASTER

O objetivo principal das pipelines de CI/CD do CodeMaster será compilar e executar a aplicação, para que seja validado que o código integrado esteja funcional. Visto isso, é necessário que a máquina que esteja executando o projeto tenha todos os componentes de software necessários para isto.

Contudo, é inviável que seja configurado todos os requisitos de software do CodeMaster em uma máquina real específica para executar estas pipelines, visto que estes requisitos podem mudar com o tempo, e é possível que seja necessário executar as pipelines em máquinas diferentes, fazendo com que esta opção tenha baixa portabilidade e viabilidade.

Para realizar isto, então, com base no que foi mostrado nos capítulos anteriores, foi criada uma imagem Docker base para o projeto (DOCKER INC. 2022e). Essa imagem Docker contém todos os requisitos de software necessários para compilar e executar o CodeMaster, e, utilizando a tecnologia Docker, ela se torna completamente portável, e pode ser utilizada em qualquer máquina que tenha o Docker instalado.

Como dito anteriormente, o Docker Hub (DOCKER INC, 2021b) é o repositório oficial para imagens Docker. Com ele, é possível guardar imagens

Docker para que elas possam ser utilizadas depois em qualquer caso de uso. Por isso, o GQS criou uma conta para o laboratório no Docker hub para armazenar essas e outras imagens que o laboratório possa ter.

Para criar esta imagem Docker base do CodeMaster, foi criado um arquivo ***Dockerfile*** com os comandos necessários para preparar o ambiente, com base no que foi indicado nas sessões anteriores.

No trecho de código a seguir, é apresentado o ***Dockerfile*** atualizado que foi utilizado para a criação da imagem Docker:

```
FROM tomcat:8-jdk8

RUN apt-get update && apt-get install -y \
   maven \
   python3-dev \
   python3-pip \
   nodejs \
   npm \
   git-all \
   ant \
   cmake \
   vim

RUN npm install -g n && n stable

RUN cd ~ && git clone https://github.com/opencv/opencv.git && \
 cd opencv && \
 git checkout 3.4.3 && \
 mkdir build && \
 cd build && \
 export ANT_HOME=/usr/share/ant && \
 cmake .. -DBUILD_SHARED_LIBS=OFF && \
 make -j8 && \
 mvn install:install-file -Dfile=bin/opencv-343.jar
-DgroupId=org.openpnp -DartifactId=opencv -Dversion=343
-Dpackaging=jar

RUN mkdir -p /srv/codemaster/backend/temp/
RUN mkdir -p /srv/codemaster/backend/projects/
RUN mkdir -p /srv/codemaster/backend/temp/copia_TM
RUN mkdir -p /srv/codemaster/backend/projects/copia_AIA

EXPOSE 8080 8801 4200 3000 8000 9999 3060
```

Esse ***Dockerfile*** possui todo o passo a passo para preparar o ambiente de execução do projeto CodeMaster, conforme foi explicado anteriormente, porém com algumas alterações:

1. Foram adicionadas quatro pastas auxiliares que o CodeMaster necessita para lidar com upload de arquivos, algo que não havia sido mapeado anteriormente.
2. Foram expostas uma série de portas do contêiner para o ambiente externo, usando o comando **EXPOSE**.

As portas do contêiner foram expostas diretamente na imagem Docker base, pois após uma análise do projeto, foi identificado que estas imagens poderiam ser usadas pelos desenvolvedores em tempo de desenvolvimento, para que eles possam rodar o projeto. Por padrão, o Docker não possibilita ao desenvolvedor acessar nenhum processo rodando dentro de um contêiner, a não ser que o processo esteja exposto ao ambiente externo através do comando **EXPOSE.** Uma explicação mais detalhada desta motivação será dada na próxima sessão.

Com o ***Dockerfile*** criado, bastou construí-lo para gerar uma imagem Docker, utilizando o comando:

```
$ docker build --rm -t gqsufsc/codemaster:latest .
```

E então foi realizado o push da imagem Docker construída para a conta do Laboratório GQS, usando o comando:

```
$ docker push gqsufsc/codemaster:latest
```

Com isso, a imagem Docker base já com todos os requisitos prontos para rodar o projeto está disponível para uso público através do seguinte link: https://hub.docker.com/r/gqsufsc/codemaster.

E essa imagem Docker pode ser facilmente referenciada em outros ***Dockerfiles*** ou pipelines de CI/CD através da sua tag **gqsufsc/codemaster:latest**.

4.1.2 CRIAÇÃO DE IMAGENS BASE PARA O GQS

Imagens Docker, quando executadas, geram containers. Estes containers podem facilmente ser utilizados pelos desenvolvedores do laboratório como máquinas virtuais, ou seja, ambientes prontos com todo o instrumental de software já instalado para que eles possam realizar tarefas e atuar nos projetos.

Com isso, foi identificada a oportunidade de, além de utilizar esta imagem Docker base do CodeMaster como ambiente para as pipelines de CI/CD, também utilizá-la como ambiente para que os desenvolvedores possam atuar e executar o projeto localmente.

Contudo, mesmo que um desenvolvedor utilize esta imagem Docker base para gerar um container Docker, e consiga rodar o CodeMaster com sucesso, ele não conseguirá acessar a aplicação rodando no container de fora dele. Isso porque,

por padrão, o Docker não permite que o ambiente externo acesse algum processo rodando dentro do container.

E esta foi a motivação para ser adicionado o comando **EXPOSE** a imagem Docker base do projeto. Com ele, é definida uma série de portas do container que são acessíveis ao ambiente externo, para que os desenvolvedores possam executar o projeto, ou até apenas uma parte dele, em uma dessas portas, e acessá-lo das suas máquinas locais.

Visto esta oportunidade, observou-se uma outra oportunidade de ter outras imagens Docker base para outros casos de uso do laboratório. Por isso, foram criados outros Dockerfiles para os demais projetos mais importantes e tecnologias atualmente mais utilizadas no GQS, sendo eles:

- **Dockerfile para o projeto App Inventor:**
  O App Inventor é um outro grande projeto do GQS, que não foi aprofundado neste trabalho, mas é muito utilizado pelo laboratório.
  Visto isso, foi identificada a oportunidade de criar uma imagem Docker base para desenvolvimento deste projeto:

```
FROM tomcat:8-jdk8

## install software dependencies
RUN apt-get update && apt-get install -y \
  python3-dev \
  python3-pip \
  git-all \
  ant \
  vim

EXPOSE 8080 8801 4200 3000 8000 9999 3060
```

- **Dockerfile para projetos de frontend:**
  O GQS possui uma quantidade grande de projetos de software que possuem interfaces web. Visto isso, foi identificada a oportunidade de ter uma imagem Docker com alguns componentes de software básicos para criar e atuar nestes projetos:

```
FROM tomcat:8-jdk8

## install software dependencies
RUN apt-get update && apt-get install -y \
  maven \
  python3-dev \
  python3-pip \
```

```
  nodejs \
  npm \
  git-all \
  vim

RUN pip install flask

## install node version manager n and change to latest version
RUN npm install -g n && n stable

RUN npm install -g @angular/cli

EXPOSE 8080 8801 4200 3000 8000 9999 3060
```

- **Dockerfile para projetos de Machine Learning:**
  O laboratório também trabalha bastante com projetos de machine learning, estes projetos utilizam alguns componentes de software específicos que são bastante complexos de instalar, como o **OpenCV**, por exemplo.
  Visto isso, também foi criada uma imagem Docker base que já possui estes componentes pré-instalados:

```
FROM python:latest

RUN apt-get update && apt-get install -y \
  python3-dev \
  python3-pip \
  git-all \
  ant \
  cmake \
  vim


RUN cd ~ && git clone https://github.com/opencv/opencv.git && \
 cd opencv && \
 git checkout 3.4.3 && \
 mkdir build && \
 cd build && \
 export ANT_HOME=/usr/share/ant && \
 cmake .. -DBUILD_SHARED_LIBS=OFF && \
 make -j8

EXPOSE 8080 8801 4200 3000 8000 9999 3060
```

Todos estes Dockerfiles foram reunidos em um repositório do laboratório GQS no GitLab, para evitar que sejam perdidos, e possam ser facilmente acessados por outros desenvolvedores no futuro.

Cada um destes Dockerfiles foram construídos em imagens Docker, que foram enviadas ao Docker Hub, na conta do laboratório GQS, e todas elas estão disponíveis para acesso público na url: https://hub.docker.com/u/gqsufsc.

Para cada imagem Docker, foi criada uma documentação indicando qual seu propósito, quais seus componentes de software, e quais as portas expostas, para que esta informação seja facilmente acessível a desenvolvedores futuros do laboratório.

Com isso, o GQS agora possui uma série de imagens base para a maioria dos seus muitos casos de uso. Os desenvolvedores podem utilizar estas imagens tanto em desenvolvimento, quanto em outras pipelines de CI/CD.

4.1.3 CRIAÇÃO DA PIPELINE DE EXEMPLO PARA O CODEMASTER

Tendo a imagem Docker base pronta e acessível através do Docker Hub, bastou implementar a pipeline de CI/CD no projeto CodeMaster. Porém, o processo de criar estas pipelines seria bastante experimental e, para evitar afetar o desenvolvimento do projeto por outros desenvolvedores, foi criado um projeto auxiliar para a implementação destas pipelines:

Figura 10: Página inicial do projeto de testes de CI/CD
(Fonte: elaborado pelo autor)

Este projeto temporário foi criado com o mesmo código do CodeMaster, e nele foi feito toda a experimentação até chegar em uma pipeline de CI/CD ideal para o projeto.

4.1.3.1 HABILITANDO CI/CD NO PROJETO

O primeiro passo para implementar a pipeline, foi habilitar a integração e deployment contínuos no projeto. Para isso, foi necessário indicar ao GitLab que os fluxos CI/CD estavam habilitados no projeto, para isso bastou acessar a configuração presente em **Settings > General > Visibility, project features, permissions** da interface do projeto no GitLab, ir até **CI/CD**, e habilitar:

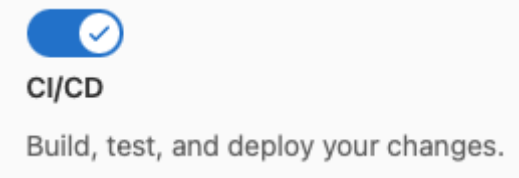


Figura 11: Interface de habilitação do CI/CD
(Fonte: elaborado pelo autor)

Com apenas esta pequena configuração, o GitLab estava configurado para tentar executar pipelines de CI/CD quando necessário no projeto.

4.1.3.2 CONFIGURANDO O RUNNER DA PIPELINE

Com o CI/CD habilitado no projeto, como dito nas seções anteriores, é necessário configurar um ***Runner*** para executar estas pipelines. Esta configuração é feita no menu **Settings > CI/CD > Runners** da interface do projeto no GitLab.

Olhando esta sessão, é possível ver que o laboratório já possui uma série de **Runners Compartilhados** prontos para serem usados nas pipelines:

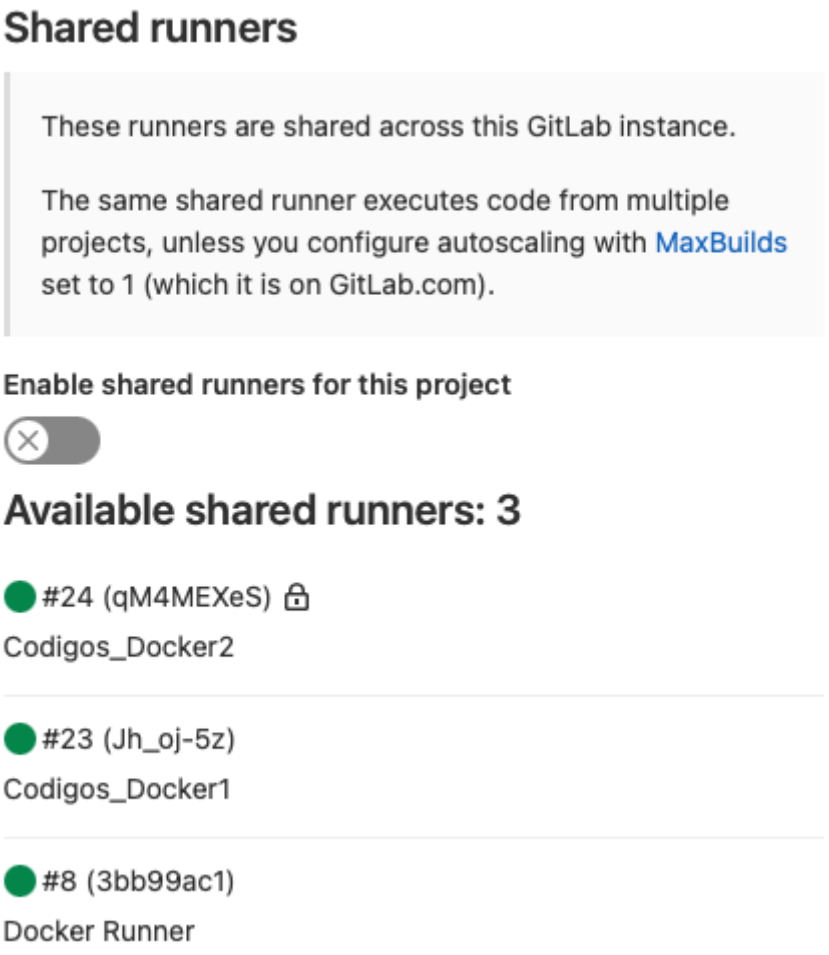


Figura 12: Interface de listagem de runners compartilhados
(Fonte: elaborado pelo autor)

Estes runners são compartilhados entre os projetos, e estão prontos para serem utilizados e aptos para executar pipelines, porém, eles não foram usados por dois motivos principais:

1. O pretexto de criar esta pipeline neste projeto é juntar todas as informações práticas necessárias para a criação do guia de CI/CD do laboratório, por isso, é preciso realizar o processo de configuração completo, incluindo a configuração de um runner do zero.
2. Um dos objetivos básicos desta pipeline, que foi mencionado anteriormente, seria fazer o *deploy* do projeto para o ambiente de desenvolvimento. Visto isso, é interessante que estas pipelines sejam executadas diretamente pela máquina onde o servidor de desenvolvimento estará rodando, e não por outras máquinas.

Por conta do ponto 2, então, o runner do projeto foi configurado em uma máquina virtual de desenvolvimento fornecida pela SeTIC, e toda a pipeline do projeto é rodada diretamente na máquina.

Esta máquina de desenvolvimento foi acessada através da máquina pessoal do aluno utilizando ***ssh***. Com isso, foi possível instalar a aplicação gitlab-runner (GITLAB B. V. 2022e) nesta máquina, e utilizar o comando ***gitlab-runner register*** para registrar a máquina como um runner para executar as pipelines do projeto.

Ao rodar este comando, algumas informações foram necessárias para configurar o runner e adicioná-lo ao projeto, sendo elas:

- **URL e token do projeto do GitLab**: Estas são as duas informações que irão relacionar o Runner a um projeto do GitLab.
  É possível achar essas informações na página **Settings > CI/CD > Runners**, na aba **Specific runners**:

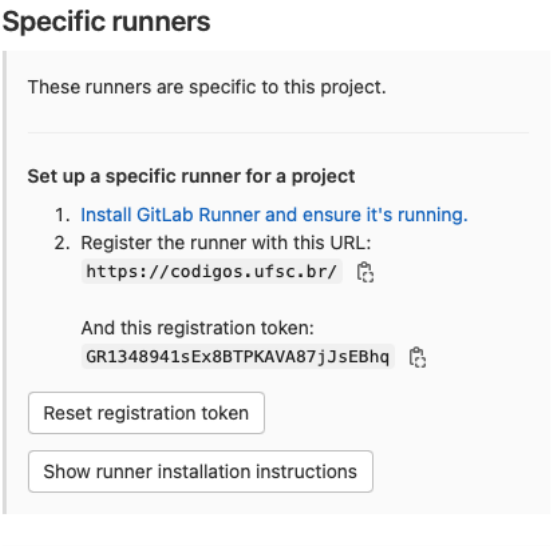


Figura 13: Interface de configuração de runners específicos
(Fonte: elaborado pelo autor)

- **executor:** Como dito anteriormente, um ***Runner*** deve possuir um executor, que indica qual a maneira que o runner irá executar a pipeline.
  Para a experimentação da pipeline do CodeMaster, foram configurados dois executores na mesma máquina de testes para o projeto, um contendo o executor **docker,** e outro contendo o executor **shell.**

O executor **docker** roda as pipelines em uma imagem Docker especificada pelo desenvolvedor, esta imagem Docker gerada é apagada ao fim da execução da pipeline, ou seja, ao fim da execução, tudo o que foi feito é apagado. Já o executor **shell** irá executar a pipeline direto na máquina, qualquer comando que for executado na pipeline, terá efeito diretamente na máquina de testes. Estes dois executores foram configurados para que possa ser realizada uma experimentação entre diferentes modos de execução de pipelines.

Após a configuração ter sido feita, os runners estavam aptos para interpretar e executar as pipelines.

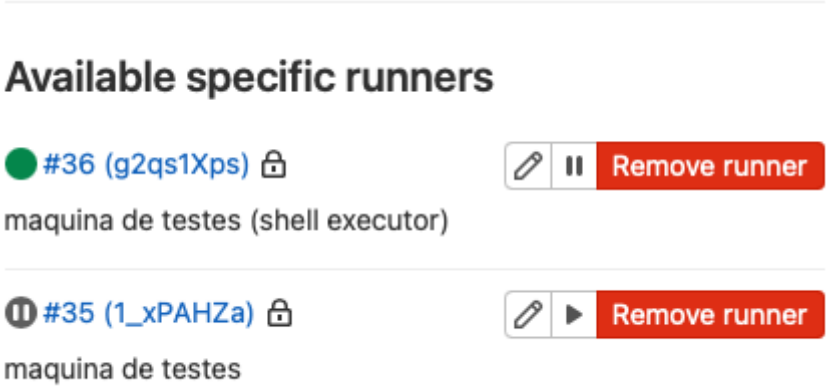


Figura 14: Lista de Runners configurados no CodeMaster
(Fonte: elaborado pelo autor)

4.1.3.3 PLANEJANDO A PIPELINE

Com os runners executados, o passo final desta configuração foi criar a pipeline de CI/CD para o CodeMaster.

Reiterando, estes são os objetivos principais desta pipeline:

- Carregamento do código fonte do CodeMaster do GitLab para a máquina de desenvolvimento;
- Construção de cada subprojeto e artefato que compõe o CodeMaster;
- *Deploy* do projeto para o ambiente de desenvolvimento.

Em outras palavras, essas pipelines devem compilar o projeto na máquina de desenvolvimento, e executar um servidor que deve ser acessível pelo ambiente externo.

Com estes objetivos em mente, a primeira tentativa foi utilizar o ***Runner*** que foi configurado com o executor **docker**, utilizando a imagem Docker base do CodeMaster criada nas sessões anteriores como ambiente para executar o projeto.

Para realizar um teste inicial, foi criado um arquivo ***gitlab-ci.yml*** contendo uma pipeline básica que roda o frontend do projeto:

```
image: gqsufsc/codemaster:latest

stages:
  - test

start test server:
  script:
  - cd codemaster-angular
  - npm install
  - ng serve
```

O gitlab-ci, ao iniciar uma pipeline de integração contínua, já carrega o código que está sendo integrado para a máquina onde o ***Runner*** está instalado, então para esta pipeline de testes, bastou navegar até o componente web do CodeMaster, o **codemaster-angular**, construí-lo, e executá-lo.

Ao final da pipeline, o servidor deveria estar rodando na porta 4200 da máquina de desenvolvimento e, como esta máquina está dentro da rede da UFSC, o frontend deveria ser acessível através do link **http://<ENDEREÇO IP>:4200** (onde **<ENDEREÇO IP>** é o ip da máquina de desenvolvimento), por qualquer máquina dentro desta mesma rede.

A pipeline estava executando com sucesso, e o servidor estava sendo levantado sem erros. Contudo, ainda não estava sendo possível acessar o projeto na URL especificada.

Após um tempo de estudo e pesquisa, foi identificado que o executor **docker** não atenderia o caso de uso do CodeMaster, pois o objetivo principal da pipeline é, ao final dela, ter um servidor de testes rodando em ambiente de desenvolvimento, e o executor **docker**, ao finalizar uma pipeline, deleta o container que foi gerado e apaga tudo o que foi feito. Por conta disso, o servidor estava sendo executado com sucesso, porém, não estava sendo **persistido** na máquina para acesso posterior.

Com isso, o executor **shell** foi o escolhido para executar as pipelines. Pelo fato do ***Runner*** estar configurado na própria máquina de desenvolvimento, bastaria a pipeline rodar e executar o projeto que, ao final da execução, o projeto estaria rodando corretamente no ambiente de desenvolvimento.

Contudo, a máquina de desenvolvimento ainda não tem todas as dependências de software necessárias para rodar o CodeMaster, e nem deveria, visto que a imagem Docker base do projeto foi criada para evitar que seja necessário instalar estas dependências de software na máquina de desenvolvimento.

Foi necessário, então, achar uma forma de utilizar a facilidade que a imagem Docker base do CodeMaster proporciona, e ainda rodar a pipeline diretamente na máquina de testes. Para isso, foi criado então alguns ***Dockerfiles*** para o projeto.

Criar ***Dockerfiles*** para o projeto é uma técnica bastante utilizada em fluxos de integração contínua atuais. Ela consiste em criar um ou mais arquivos ***Dockerfile*** na raiz do projeto, que utilizam alguma imagem Docker como base, e possuem toda a lógica para construir e executar o projeto.

Então, quando estes ***Dockerfiles*** são transformados em imagens e executados, eles executam o projeto em questão. Desta maneira, qualquer máquina pode executar o projeto, bastando ter o Docker instalado na máquina em questão, e executar os ***Dockerfiles*** do projeto.

No contexto do CodeMaster, foi necessário criar três dockerfiles diferentes:

- **Dockerfile.backend:** Que constrói e executa todos os componentes de backend do projeto.
- **Dockerfile.frontend:** Que constrói e executa os componentes de frontend do projeto.
- **Dockerfile.database:** Que inicia um banco de dados SQL de testes para ser utilizado pela aplicação que está executando.

Desta forma, o funcionamento planejado se dá desta maneira:

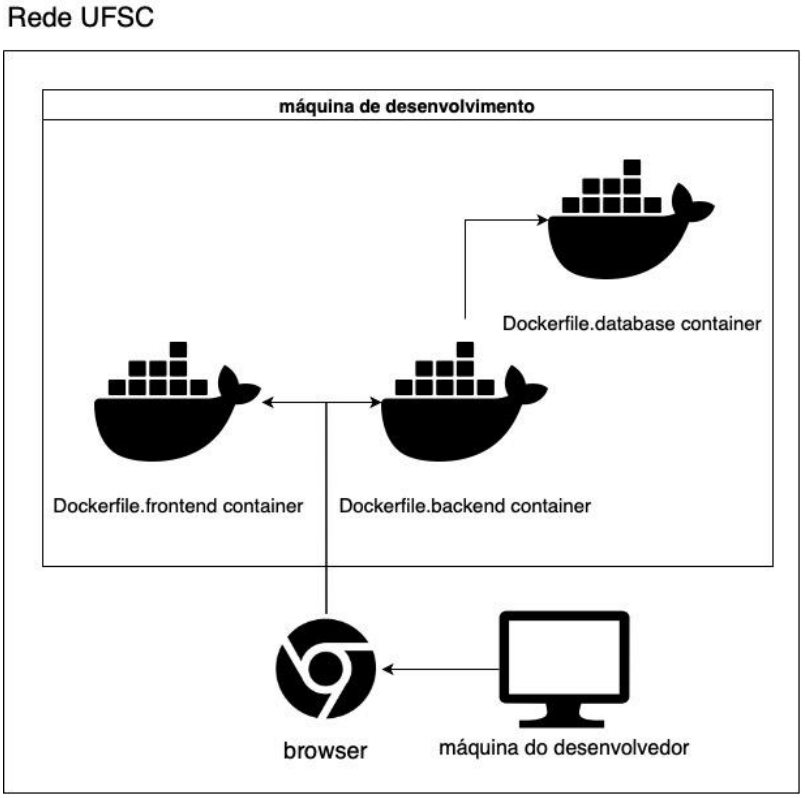


Figura 15: Diagrama de funcionamento do servidor de desenvolvimento
(Fonte: elaborado pelo autor)

A máquina de desenvolvimento estará rodando os containers de frontend, backend e de banco de dados, para o CodeMaster. Os containers de backend e frontend estarão expostos para o ambiente externo, e o container de backend irá chamar o container de banco de dados. A máquina do desenvolvedor, e a máquina de desenvolvimento, estarão na mesma rede, que é a rede fornecida pela UFSC. Desta forma, o browser da máquina do desenvolvedor poderá acessar tanto o backend quando o frontend, que estarão rodando em portas específicas da máquina de desenvolvimento.

Porém, este planejamento ainda possui uma falha crucial. Por padrão, os containers Docker não podem acessar processos rodando em outros containers, mesmo que estes processos estejam expostos ao ambiente externo (Tom Donohue 2022).

Quando é exposta uma porta do container ao ambiente externo, ao rodar o container, para que esta porta seja realmente acessível, é preciso mapear esta porta do container a uma porta da máquina que o está rodando.

Desta forma, o que é realmente acessado, não é o processo rodando no container, e sim um processo rodando na máquina onde o container está inserido, que faz uma referência ao processo rodando no container. O diagrama da Figura 16 indica um pouco melhor essa conexão, usando o caso de uso do CodeMaster como exemplo:

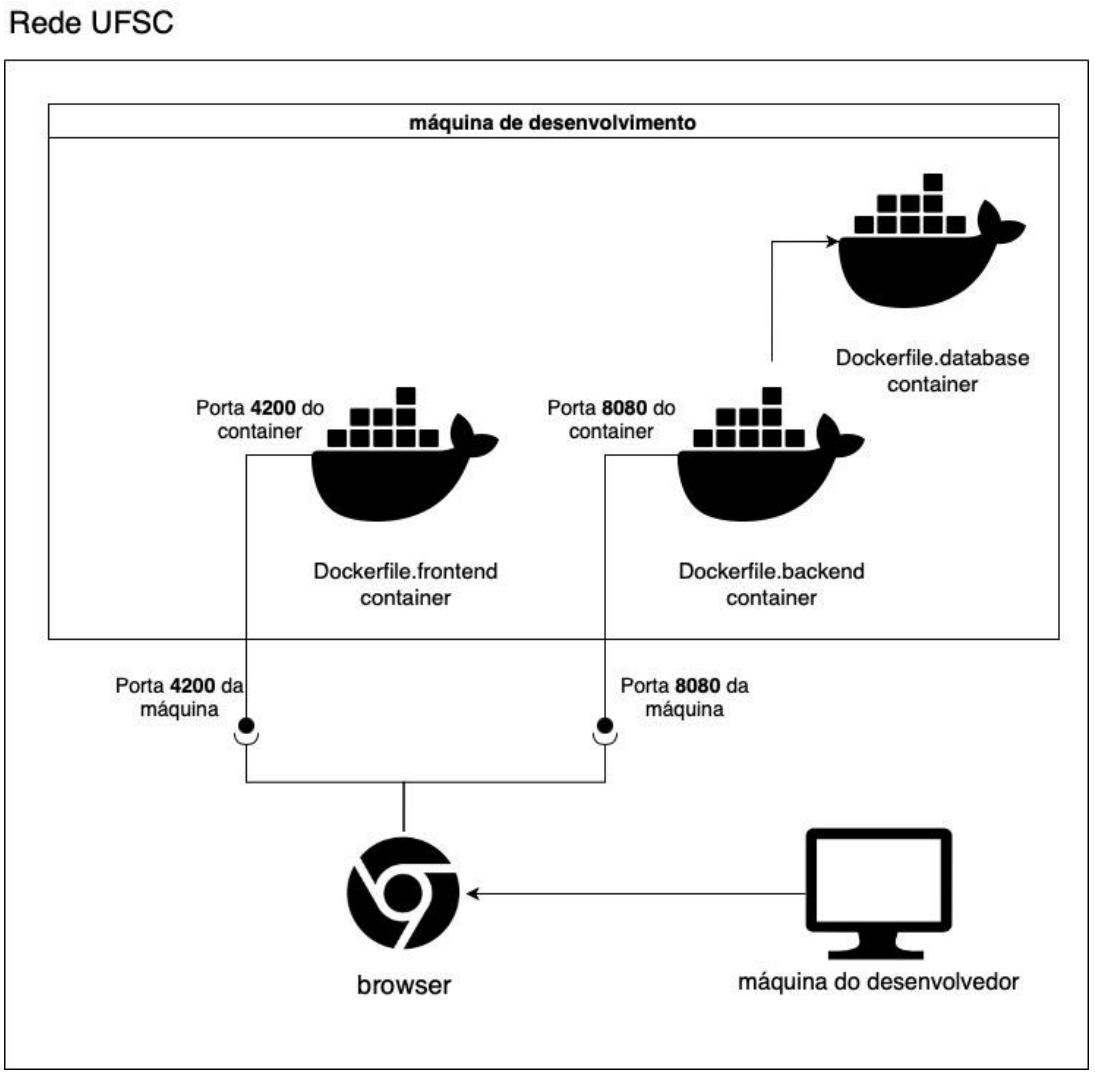


Figura 16: Diagrama de funcionamento do servidor de desenvolvimento com mapeamento de portas
(Fonte: elaborado pelo autor)

Na situação real, como mostrado no diagrama, a interface web do CodeMaster estará rodando na porta 4200 do container de frontend, que é mapeada para a porta 4200 da máquina de desenvolvimento, que é acessível pelo ambiente externo. O mesmo vale para o componente de backend, que estará rodando na porta 8080 do container, e será mapeado para a porta 8080 da máquina de desenvolvimento.

Isso é feito pois um processo rodando em um container não pode ser acessado diretamente pelo ambiente externo, por padrão, o Docker bloqueia este acesso. Esta é uma regra que vale também para acessos entre containers, ou seja, um container não pode acessar um processo rodando em outro. Porém, na estrutura planejada, é necessário que o container do backend acesse o container do banco de dados. Para isso, foram utilizadas as redes do Docker (DOCKER INC. 2022f).

Com essa funcionalidade, é possível que um container acesse outro, desde que eles estejam na mesma **rede** do Docker. O diagrama a seguir demonstra o diagrama final do planejamento do servidor de testes:

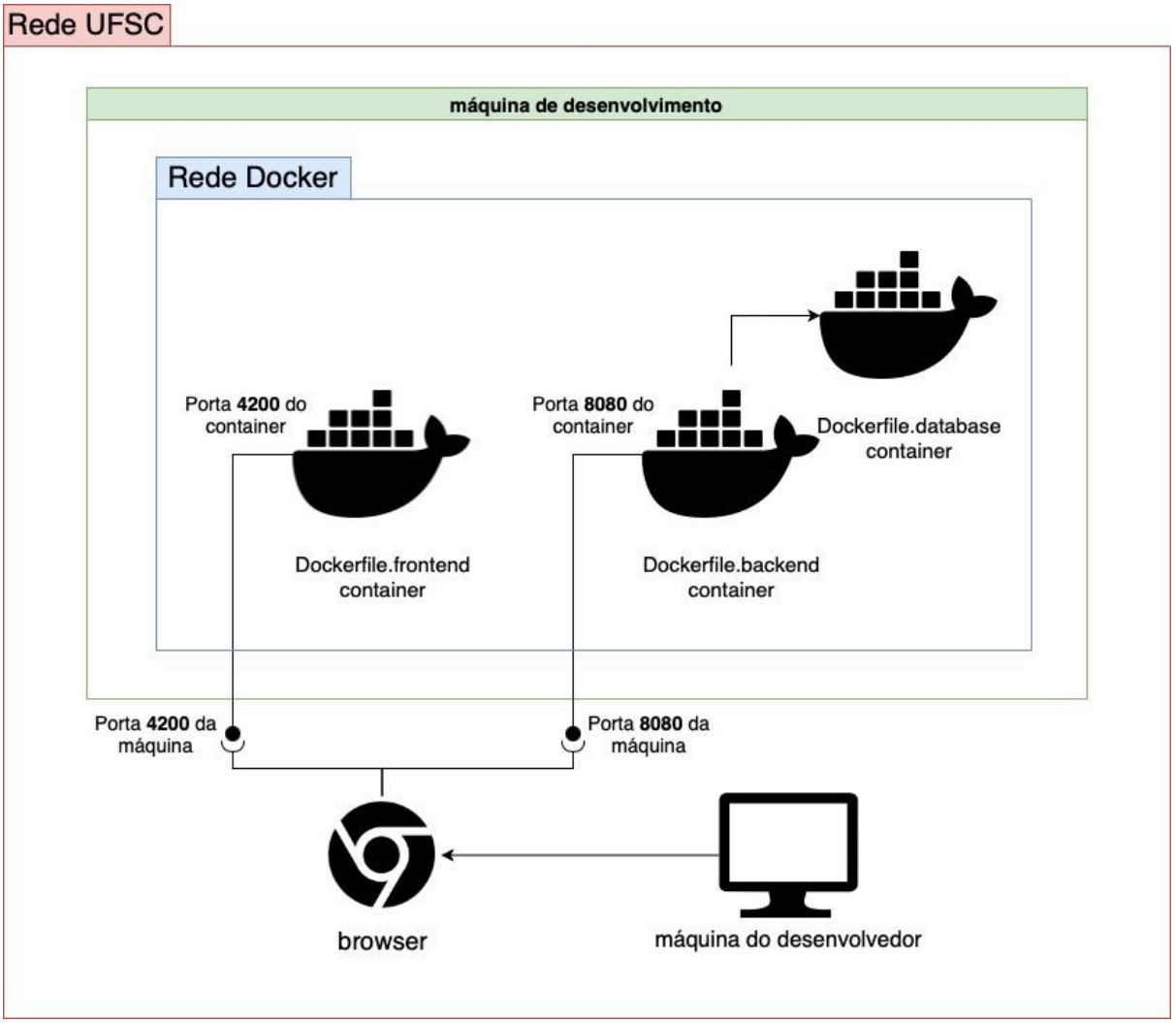


Figura 17: Diagrama final de funcionamento do servidor de desenvolvimento
(Fonte: elaborado pelo autor)

No esquema apresentado, todos os três containers Docker estarão inseridos em uma mesma rede Docker, possibilitando que eles sejam acessíveis entre si. A aplicação possui dois componentes, um de front-end, e outro de back-end, rodando nas portas **4200** e **8080** da máquina de desenvolvimento, respectivamente.

Desta forma, a aplicação será acessível por qualquer máquina presente na rede da UFSC, através dos links **http://<ENDEREÇO IP>:4200** (para o componente web) e **http://<ENDEREÇO IP>:8080** (para o componente de backend), onde **<ENDEREÇO IP>** é o ip da máquina virtual de desenvolvimento.

4.1.3.4 CRIANDO OS DOCKERFILES

Com a estrutura do servidor de testes planejada, o passo seguinte foi implementar os ***Dockerfiles*** que executarão o projeto. Como dito anteriormente, são 3 ***Dockerfiles*** que compõem a lógica de execução do CodeMaster.

4.1.3.4.1 DOCKERFILE DE BACKEND

O ***Dockerfile.backend*** é responsável por construir e executar todos os componentes de backend do projeto, e está presente no trecho de código a seguir, com seus passos numerados para facilitar a explicação posterior:

```
# 1
FROM gqsufsc/codemaster:latest

# 2
COPY / /home/app

# 3
RUN pip install -i https://test.pypi.org/simple/ creassessment==1.3.6.4
RUN pip install -r /home/app/Appsthetics/requirements.txt -f
https://download.pytorch.org/whl/torch_stable.html
RUN python3 /home/app/Appsthetics/server.py &

# 4
RUN mvn -f /home/app/Commons/pom.xml clean package
RUN mvn install:install-file
-Dfile=/home/app/Commons/target/Commons-2.0-SNAPSHOT.jar
-DgroupId=br.ufsc.cne.codemaster -DartifactId=Commons -Dversion=2.0-SNAPSHOT
-Dpackaging=jar

# 5
RUN mvn -f /home/app/codemaster-apirest/pom.xml clean package -DskipTests
RUN mv /home/app/codemaster-apirest/target/codemaster-spring-2.0-SNAPSHOT.war
/usr/local/tomcat/webapps/codemaster.war

# 6
RUN mvn -f /home/app/RESTGrader/pom.xml clean package -DskipTests
RUN mv /home/app/RESTGrader/target/RESTGrader-2.0-SNAPSHOT.war
/usr/local/tomcat/webapps/RESTGrader.war

# 7
EXPOSE 8080
ENV JAVA_OPTS="-Djava.library.path=/root/opencv/build/lib"
CMD ["catalina.sh", "run"]
```

Esse Dockerfile executa as seguintes ações, nesta ordem:

- **#1:** Buscar a imagem Docker base do CodeMaster (DOCKER INC. 2022e) no docker-hub, usando a tag especificada anteriormente;
- **#2:** Copiar os arquivos do diretório do ***Dockerfile*** para dentro da imagem Docker;
- **#3:** Instalar as dependências do componente AppSthetics e executá-lo;
- **#4:** Construir o componente **Commons** e adicioná-lo ao tomcat;
- **#5:** Construir o componente **codemaster-apirest** e adicioná-lo ao tomcat;
- **#6:** Construir o componente **RESTGrader** e adicioná-lo ao tomcat;
- **#7:** Expor a porta 8080 da imagem Docker e definir o comando ***catalina.sh run*** como ponto de partida do container.

Quando o ***Dockerfile*** for construído, ele irá gerar uma imagem Docker com todos os componentes de backend do projeto CodeMaster construídos e prontos para a execução.

Quando esta imagem Docker for executada, será executado o comando ***catalina.sh run***, assim iniciando o servidor de aplicações Tomcat e executando os componentes **codemaster-apirest** e **RESTGrader**, que estarão rodando em um container Docker.

4.1.3.4.2 DOCKERFILE DE BANCO DE DADOS

O ***Dockerfile*.database** é responsável por preparar e executar um banco de dados de teste para o ambiente de desenvolvimento do CodeMaster. Ele usa algumas funcionalidades da imagem Docker Docker oficial do MySQL (DOCKER INC. 2022c) para facilitar este processo.

A seguir, é mostrado o trecho de código que foi utilizado para criar o arquivo, já com seus passos numerados para facilitar a explicação posterior.

```
# 1
FROM mysql

# 2
ENV MYSQL_DATABASE bd_code_master

# 3
ENV MYSQL_ROOT_PASSWORD root

# 4
COPY ./codemaster-apirest/db/ /docker-entrypoint-initdb.d/
```

- **#1:** Buscar a imagem Docker base do MySQL (DOCKER INC. 2022c) no docker-hub;

- **#2:** Define a variável **MYSQL_DATABASE**, que mostrará ao mysql que o banco **bd_code_master** deverá ser usado.
- **#3:** Define a variável **MYSQL_ROOT_PASSWORD**, que definirá a senha de acesso root ao banco.
- **#4:** Copia todos os arquivos dentro da pasta ***./codemaster-apirest/db*** para dentro do diretório ***/docker-entrypoint-init.d/***.

Ao construir o Dockerfile, ele irá gerar uma imagem Docker que, quando executada, irá subir uma instância de um banco de dados mysql, e ler quaisquer arquivos com a extensão ***.sql*** presentes no diretório ***/docker-entrypoint-init.d/*** no banco de dados especificado.

Isso permite com que os desenvolvedores definam scripts a serem executados ao iniciar o banco de teste, bastando colocá-los no diretório ***./codemaster-apirest/db*** que, quando o ***Dockerfile*** for executado, serão carregados para o diretório ***/docker-entrypoint-init.d/***.

Sempre que este Dockerfile for executado novamente, este servidor MySQL será completamente reiniciado e os scripts serão executados novamente, visto isso, qualquer dado persistido no banco de testes será apagado. Caberá aos pesquisadores do laboratório adicionar aos scripts quaisquer dados de teste necessários.

4.1.3.4.3 DOCKERFILE DE FRONTEND

O ***Dockerfile.frontend*** é responsável por rodar o componente web do projeto. Este componente utiliza apenas o NodeJS e o Angular como dependências para rodá-lo, por isso, ao invés de utilizar a imagem Docker base do CodeMaster, foi utilizada a imagem Docker base de frontend do GQS (DOCKER INC. 2022c), por ela ser mais leve e possuir todo o necessário para rodar este componente.

```
# 1
FROM gqsufsc/frontend:latest

# 2
COPY / /home/app

# 3
WORKDIR /home/app/codemaster-angular
RUN npm install

# 4
EXPOSE 4200
CMD [ "ng", "serve", "--host=0.0.0.0", "--port=4200"]
```

- **#1:** Buscar a imagem Docker base de frontend do laboratório no docker-hub, usando a tag especificada anteriormente;
- **#2:** Copiar os arquivos do diretório do ***Dockerfile*** para dentro da imagem Docker;
- **#3:** Navegar para o diretório do componente web ***codemaster-angular*** e instalar o projeto;
- **#4:** Expor a porta 4200 do container e definir o comando ***ng serve --host=0.0.0.0 --port=4200*** como ponto de partida do projeto.

Quando esse arquivo for construído, ele instalará todo o componente web do CodeMaster em uma imagem Docker que, quando executada, irá executar o componente ***codemaster-angular***.

4.1.3.5 CRIANDO UM AMBIENTE DE TESTES

Como informado, a pipeline planejada irá subir um servidor em um ambiente de desenvolvimento, que estará acessível a qualquer máquina na rede UFSC através da url **http://<ENDEREÇO IP>:4200**.

Para fins de documentação, é interessante que esteja explicitamente identificado em alguma localização de fácil acesso que este servidor está rodando. Para isso, é possível utilizar a funcionalidade de ambientes do GitLab (GITLAB B. V. 2022f).

Utilizando esta funcionalidade, é possível especificar explicitamente quais servidores estão rodando para a aplicação, identificando de qual ambiente se trata, e qual a url que o ambiente pode ser acessado. Esta configuração de ambientes pode ser acessada através do menu ***Deployments > Environments*** da página inicial do projeto no GitLab.

No contexto do projeto, foi criado o ambiente ***staging***, e indicado que ele estará rodando na url **http://<ENDEREÇO IP>:4200**.

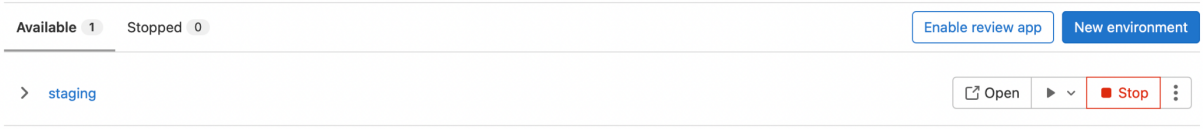


Figura 18: Ambiente de staging criado no GitLab
(Fonte: elaborado pelo autor)

É válido ressaltar que este processo é feito apenas para fins de documentação. O processo de deploy para este ambiente deverá ser feito pela pipeline de CI/CD, algo que será mostrado na seção a seguir.

4.1.3.6 IMPLEMENTANDO A PIPELINE

Com o funcionamento do servidor planejado, a pipeline de CI/CD pode ser criada sem muitos problemas, bastando utilizar a ferramenta Docker para que ela funcione corretamente.

O trecho de código a seguir representa o arquivo ***.gitlab-ci.yml*** que foi criado para implementar as pipelines do projeto CodeMaster:

```
stages:
  - build
  - test
  - stop_test

build database:
  stage: build
  script:
    - sudo docker build -f Dockerfile.database --rm -t codemaster-database .
build frontend:
  stage: build
  script:
    - sudo docker build -f Dockerfile.frontend --rm -t codemaster-frontend .
build backend:
  stage: build
  script:
    - sudo docker build -f Dockerfile.backend --rm -t codemaster-backend .

start test server:
  only:
    - dev
  stage: test
  before_script:
    - sudo docker rm -f codemaster_db || true
    - sudo docker rm -f codemaster_back || true
    - sudo docker rm -f codemaster_front || true
    - sudo docker network rm -f codemaster || true
  script:
    - sudo docker network create codemaster
    - sudo docker run --network codemaster --name codemaster_db -d
      -p 3306:3306 codemaster-database
    - sudo docker run --network codemaster --name codemaster_back -d
      -p 8080:8080 codemaster-backend
    - sudo docker run --network codemaster --name codemaster_front -d
      -p 4200:4200 codemaster-frontend
  environment:
      name: staging
      url: http://150.162.6.91:4200

stop test server:
```

```
stage: stop_test
script:
  - sudo docker rm -f codemaster_db || true
  - sudo docker rm -f codemaster_back || true
  - sudo docker rm -f codemaster_front || true
  - sudo docker network rm -f codemaster || true
when: manual
environment:
  name: staging
  action: stop
```

Esta pipeline é composta pelos seguintes ***stages*** e ***steps:***

- *Stage* **build**, que é responsável por construir a aplicação, com os steps:
    - **build database:** Constrói o arquivo ***Dockerfile.database*** para gerar a imagem Docker de banco de dados do projeto;
    - **build frontend:** Constrói o arquivo ***Dockerfile.frontend*** para gerar a imagem Docker do componente web do porjeto;
    - **build backend:** Constrói o arquivo ***Dockerfile.backend*** para gerar a imagem Docker dos componentes de back-end do projeto.

- *Stage* **test,** que é responsável por iniciar o servidor de testes, com o step:
    - **start test server:** Inicia a rede Docker e executa os as imagens na rede criada, gerando containers Docker, expondo na máquina de testes as rotas especificadas.
      Este step utiliza o atributo ***before_script*** para definir um script que será gerado antes do step começar, que remove qualquer contêiner que já esteja rodando na máquina.
      Este step também utiliza o atributo ***environment*** para indicar que ele sobe o servidor no ambiente de ***staging***, na url especificada.
      É utilizado também o atributo ***only*** para especificar que este fluxo deverá rodar apenas para a ***branch*** dev do CodeMaster. Esta branch é destinada ao código de desenvolvimento, e é apenas quando for integrado algo nela, que o servidor de desenvolvimento será executado.

- *Stage* **stop_test**, que é responsável por parar o servidor de testes, com o step:
    - **stop test server:** Remove os containers Docker do CodeMaster que estiverem rodando na máquina para encerrar o servidor de testes.
      Este step também utiliza o atributo ***environment***, com a ***action stop***, para indicar que ele encerra o servidor no ambiente de ***staging***.
      Ele também utiliza o atributo ***when*** para indicar que o job é rodado apenas manualmente.

Ao criar o arquivo ***gitlab-ci.yml*** e integrá-lo ao código, a pipeline é executada, e seu resultado pode ser verificado na sessão ***CI/CD > Pipelines*** na página do projeto no GitLab:

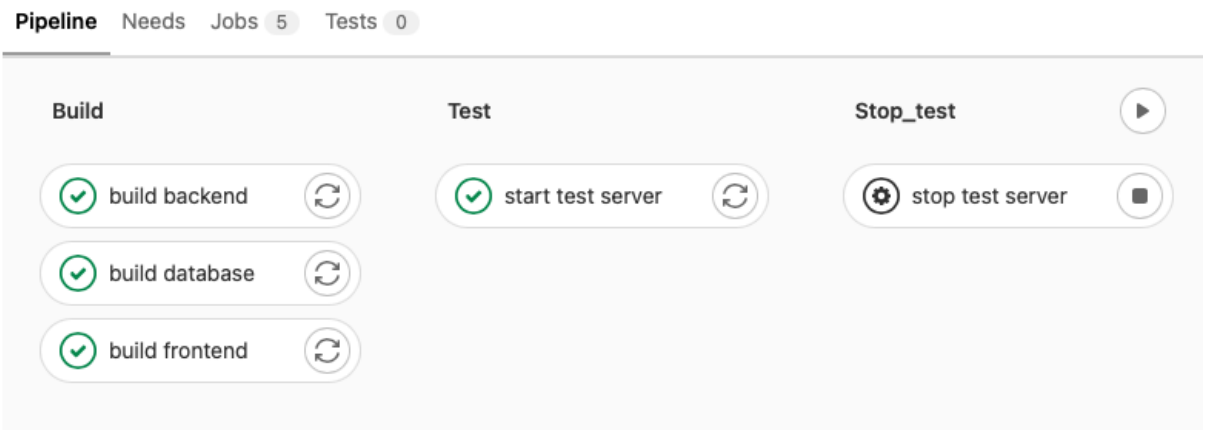


Figura 19: Resultado da pipeline de CI/CD
(Fonte: elaborado pelo autor)

Como é possível ver na figura 19, os ***stages*** são executados na mesma ordem que foi definida no atributo ***stages*** do arquivo pipeline. Pode-se verificar também que o ***step stop test server*** está com um ícone diferente, que indica que ele deve ser executado manualmente.

Quanto ao ambiente de staging que foi criado na seção anterior, graças às configurações feitas na pipeline, ele agora possui informações do seu último deploy:

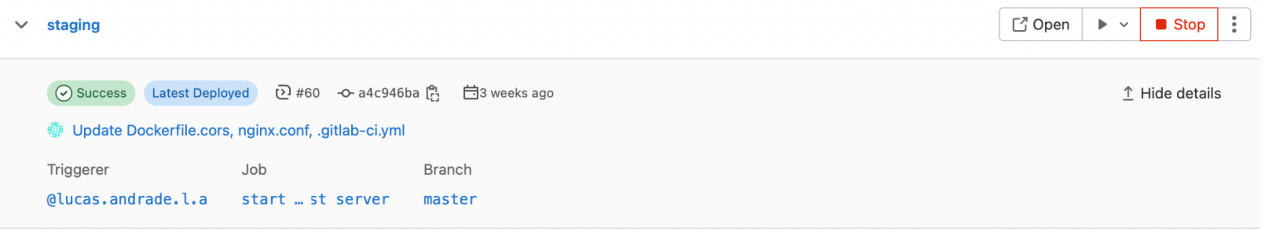


Figura 20: Ambiente de staging pronto para uso
(Fonte: elaborado pelo autor)

Como é possível ver na Figura 20, o ambiente indica várias informações qual o commit que encadeou o deploy, se a pipeline rodou com sucesso, etc.

Ao clicar no botão ***Open***, o desenvolvedor será redirecionado para a url especificada na pipeline, para que ele possa acessar o ambiente. Também, ao clicar no ícone ▸, é possível rodar o step manual que foi definido na pipeline para para o servidor de testes., o ***stop test server***:

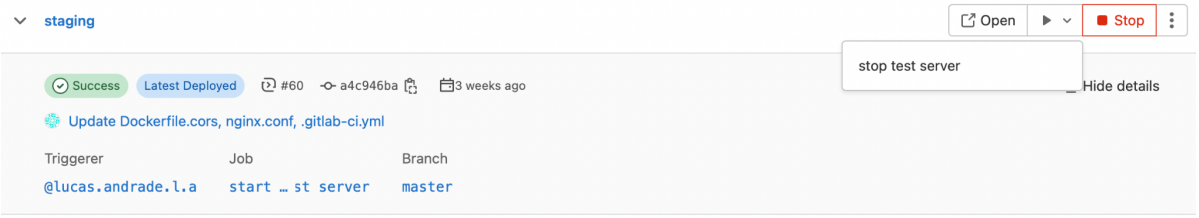


Figura 21: Interface para parar o servidor de testes
(Fonte: elaborado pelo autor)

Com isso, o servidor de testes estará rodando com sucesso na máquina de desenvolvimento do laboratório, utilizando todas as funcionalidades e facilidades do Docker, e tendo uma documentação de ambientes bem definida.

4.1.3.7 CRIAÇÃO DE UM GUIA PARA O CODEMASTER

Como dito anteriormente, o CodeMaster é uma aplicação bastante complexa, possuindo vários componentes e requisitos de software diferentes. Isso implica que montar um ambiente para desenvolvimento desta aplicação muitas vezes é uma tarefa bastante complicada e demorada.

Visto isso, percebeu-se a oportunidade de utilizar os Dockerfiles e o fluxo de CI/CD criado para facilitar esse processo de desenvolvimento. Visto que os Dockerfiles criados possuem todo o instrumental e a lógica necessária para executar a aplicação.

Por isso, foi criado um guia para desenvolvimento do projeto CodeMaster, com tudo o que o desenvolvedor precisa saber para executar e trabalhar na aplicação.

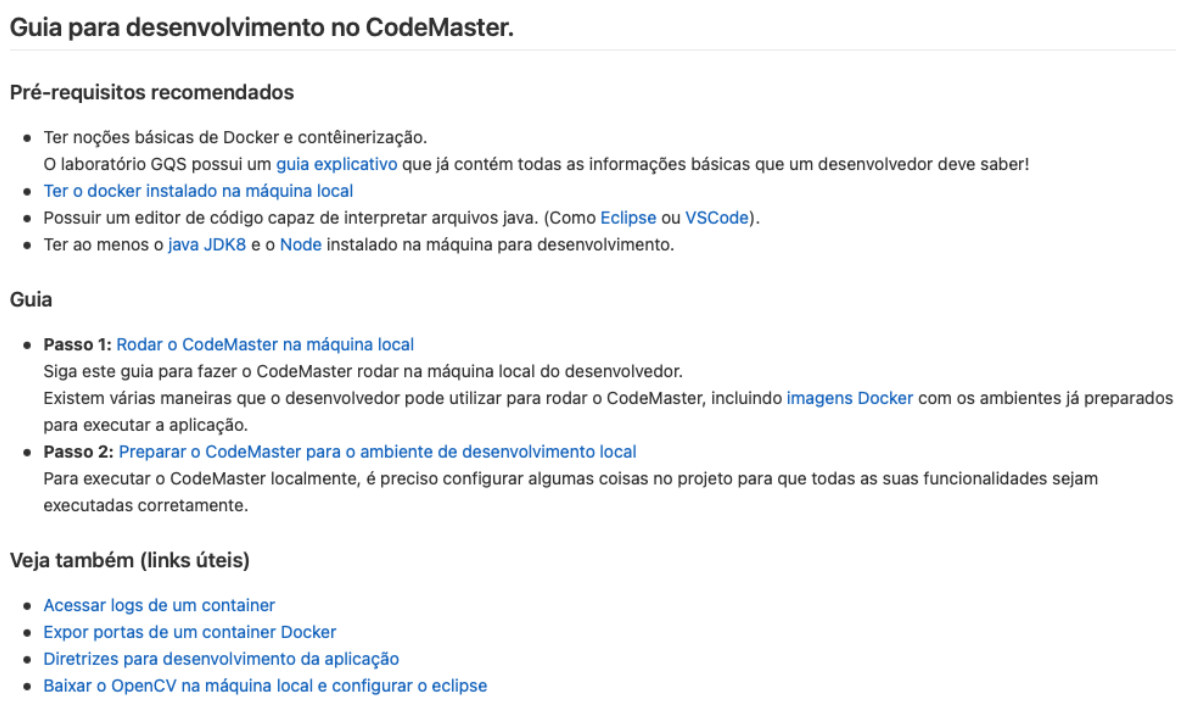


Figura 22: Página inicial do guia de desenvolvimento do CodeMaster
(Fonte: elaborado pelo autor)

Este guia foi escrito e colocado na wiki do projeto na página inicial do mesmo no GitLab, para que seja facilmente acessível pelos desenvolvedores da aplicação.

Antes de ser apresentado o guia propriamente dito, é mostrado uma sessão com alguns pré-requisitos recomendados para que o desenvolvedor consiga utilizá-lo com sucesso.

O guia é dividido em dois passos:

1. **Rodar o CodeMaster na máquina local**
   Neste passo é mostrado como rodar o projeto CodeMaster na máquina local, Onde são apresentados três possíveis jeitos de rodá-lo:
   a. **Rodar o projeto usando os Dockerfiles**
      Explica ao desenvolvedor como usar os arquivos ***Dockerfile*** da raiz do projeto para rodar a aplicação no ambiente local, se baseando no fluxo de execução da pipeline de CI/CD.

   b. **Rodar o projeto usando a imagem Docker base**
      Explica ao desenvolvedor como rodar o projeto utilizando a imagem Docker base do CodeMaster disponível no docker-hub (DOCKER INC. 2022e).
      Imagens Docker, quando executadas e transformadas em containers são, essencialmente, máquinas virtuais.
      A imagem Docker base do CodeMaster pode facilmente ser utilizada como uma máquina virtual contendo todos os requisitos de software necessários para rodar a aplicação.

Esta seção possui um passo a passo que indica ao desenvolvedor como baixar esta imagem Docker, executá-la, e utilizá-la como um ambiente de desenvolvimento.

c. **Rodar o projeto usando a máquina local**
   Esta é uma breve seção, mas sua presença é importante no guia.
   Como recurso final, o desenvolvedor pode também baixar todos os recursos de software em sua máquina e executar o projeto.
   Esta seção indica brevemente onde o desenvolvedor pode encontrar a lista de requisitos de software do projeto.

2. **Preparar o CodeMaster para o ambiente local**
   O CodeMaster possui algumas configurações que o desenvolvedor precisa realizar para que ele possa ser executado corretamente.
   Como apontamentos da API, apontamentos do banco de dados, variáveis de ambiente de arquivos temporários, etc.
   Esta seção indica exatamente quais são essas configurações necessárias e por que elas devem ser feitas. Indica também qual a configuração exata que o desenvolvedor deverá fazer, dependendo de qual opção ele escolheu para rodar o projeto no passo 1.

Seguindo estes dois passos, o desenvolvedor deverá ter o projeto rodando em sua máquina local sem muitas complicações. E também deverá saber mais sobre como o projeto funciona e qual a sua estrutura.

Ao final da página, após a apresentação do guia, é apresentado alguns links úteis para o desenvolvedor:

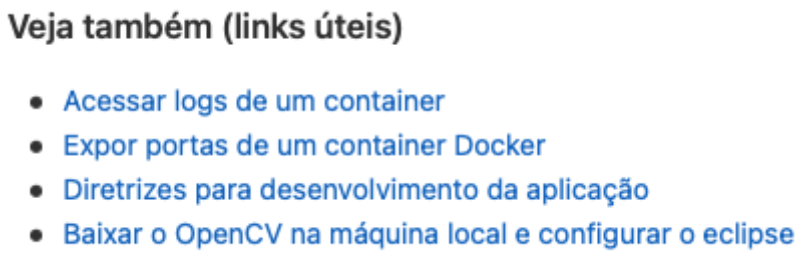


Figura 23: Links úteis do guia de desenvolvimento do CodeMaster
(Fonte: elaborado pelo autor)

Estes links contém algumas informações extras a respeito do Docker que podem ser úteis ao desenvolvedor. Além disso, possuem também alguns dos tutoriais que já existiam anteriormente para o CodeMaster, como por exemplo o tutorial de diretrizes para o desenvolvimento da aplicação, e o guia de instalação do OpenCV e configuração da IDE Eclipse.

Estes tutoriais já existentes foram colocados junto ao guia, para que esta

informação não venha a se perder, visto que é possível que estes guias também sejam úteis para algum desenvolvedor.

4.1.3.8 CRIAÇÃO DE UM GUIA PARA O SERVIDOR DE TESTES

Visto que o projeto CodeMaster não possui uma funcionalidade que permite criar variáveis de ambiente de maneira automática, para que o servidor de testes levante corretamente, algumas configurações precisam ser feitas.

Por isso, foi identificada a necessidade de criar um pequeno guia que indique aos desenvolvedores quais são estas configurações, e como lidar de maneira geral com o ambiente de desenvolvimento do CodeMaster.

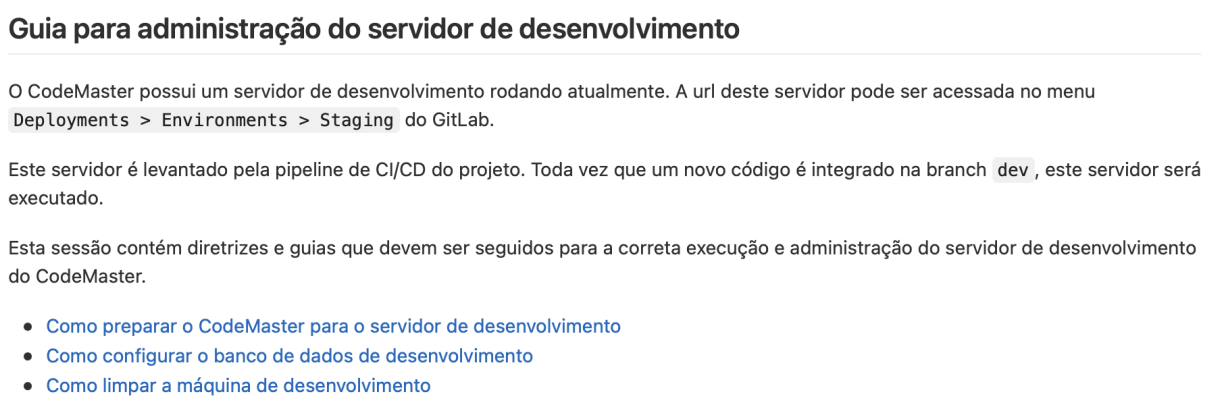


Figura 24: Guia do servidor de desenvolvimento do CodeMaster
(Fonte: elaborado pelo autor)

Este guia foi colocado junto ao guia de desenvolvimento do CodeMaster, na ***wiki*** do projeto presente na página do CodeMaster no GitLab.

O guia inicia com algumas informações gerais sobre o funcionamento do servidor, como por exemplo: como acessá-lo, como ele é levantado, e em que momento sua execução acontece.

Então, são apresentados três tópicos principais:

1. **Como preparar o CodeMaster para o servidor de desenvolvimento**
   Para que o servidor execute corretamente, os apontamentos e variáveis de ambiente do projeto precisam estar corretamente configurados.
   Esta seção se baseia no que foi explicado no tópico ***"Preparar o CodeMaster para o ambiente local"*** do guia do CodeMaster para indicar quais configurações exatas precisam ser feitas para que o servidor rode com sucesso.
   Nela, também são indicadas boas práticas para que essas mudanças sejam facilmente distinguíveis, a fim de evitar que sejam integradas ao ambiente de produção.

2. **Como configurar o banco de dados de desenvolvimento**
   Sempre que o servidor de desenvolvimento sobe, é executado também um

banco de dados de desenvolvimento.
Como dito anteriormente, este banco de dados é levantado ao construir o arquivo Dockerfile.database que agora se encontra na raiz do projeto e, ao iniciar o banco, este Dockerfile irá rodar qualquer script sql (arquivo com a extensão .sql) presente no diretório ***/codemaster-apirest/db*** do projeto.
Esta seção indica como estes scripts são executados, e define boas práticas para a criação dos mesmos, a fim de criar um padrão de criação destes scripts para os desenvolvedores do laboratório, tornando este trabalho mais limpo e organizado.

3. **Como limpar a máquina de desenvolvimento**
   Por mais que a máquina de desenvolvimento possua bastante espaço livre para a aplicação, é relativamente comum que acabe o espaço da máquina, impossibilitando a execução do fluxo de CI/CD.
   Isso porque o Docker é uma ferramenta que comumente acumula bastante "lixo" na máquina, como arquivos de cache, imagens não usadas, entre outras coisas.
   O Docker faz isso a fim de otimizar o processamento e carregamento de imagens, porém, em contextos onde a máquina não possui um grande volume de espaço, é necessário rodar uma limpeza recorrente nestes arquivos.
   Esta seção indica aos desenvolvedores qual o procedimento para realizar esta limpeza.

4.2 CRIAÇÃO DO GUIA DE CI/CD

Com o processo exemplo de CI/CD criado no projeto CodeMaster, todas as informações práticas necessárias foram coletadas para a criação de um guia completo de CI/CD para o GQS. Este guia foi criado na ***wiki*** da página inicial do laboratório GQS no GitLab UFSC: https://codigos.ufsc.br/gqs.

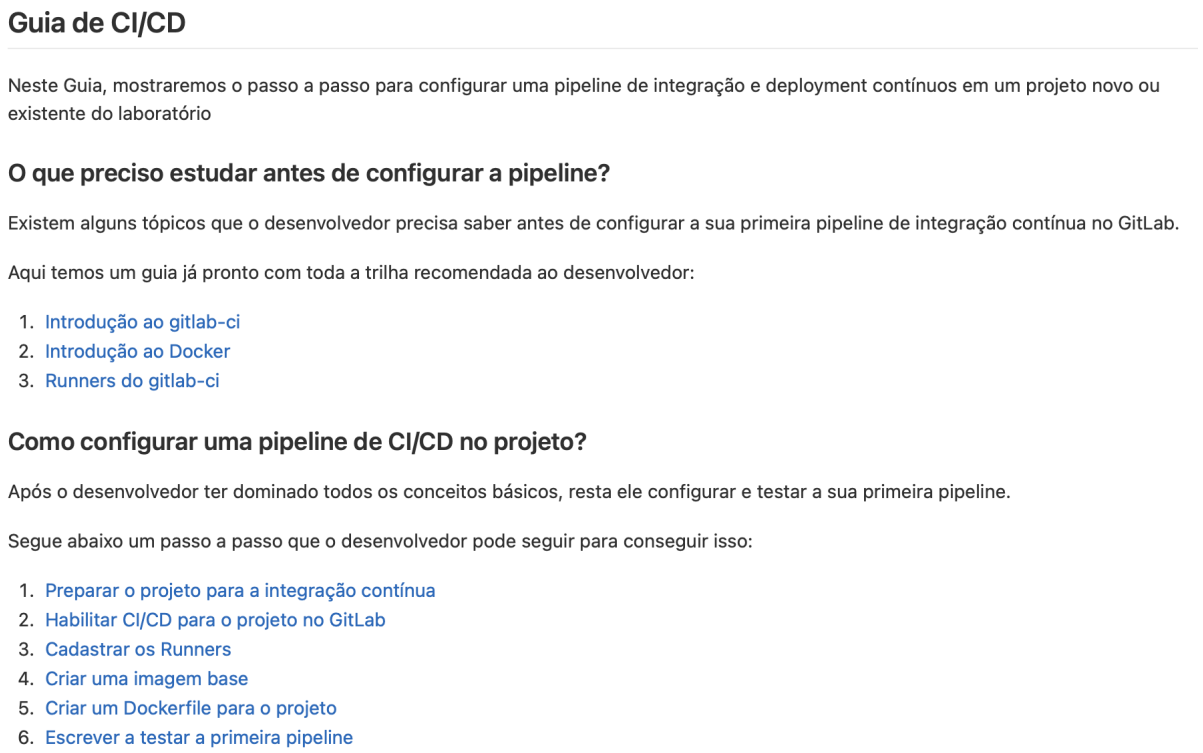


Figura 25: Página inicial do guia de CI/CD do GQS
(Fonte: elaborado pelo autor)

O objetivo principal deste guia é trazer as informações necessárias ao desenvolvedor de maneira amigável, de forma que qualquer membro do grupo GQS possa entender como as tecnologias utilizadas pelo ***gitlab-ci*** funcionam, e criar suas próprias pipelines de integração contínua usando a ferramenta.

Nas seções seguintes, serão apresentados cada um dos tópicos e subtópicos do guia.

4.2.1 TÓPICO 1: O QUE ESTUDAR

Antes de começar a configurar um guia de CI/CD, existem alguns conceitos básicos que o desenvolvedor precisa dominar ou, pelo menos, entender (Figura 26).

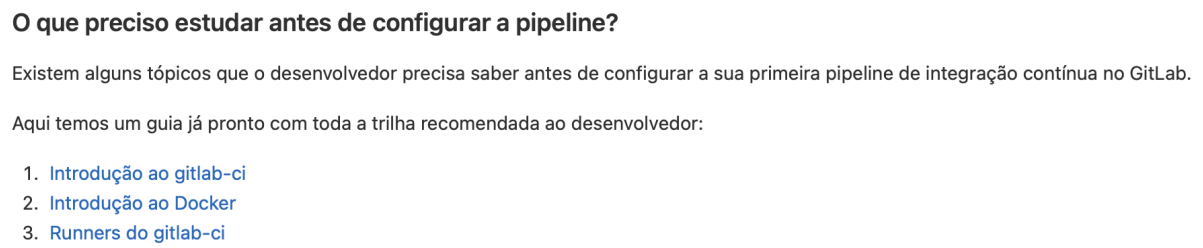


Figura 26: Tópico 1: o que estudar, do guia de CI/CD do GQS
(Fonte: elaborado pelo autor)

Este tópico apresenta uma trilha de guias informativos introdutórios que o desenvolvedor deverá seguir a fim de obter as informações teóricas necessárias para configurar a sua primeira pipeline de CI/CD.

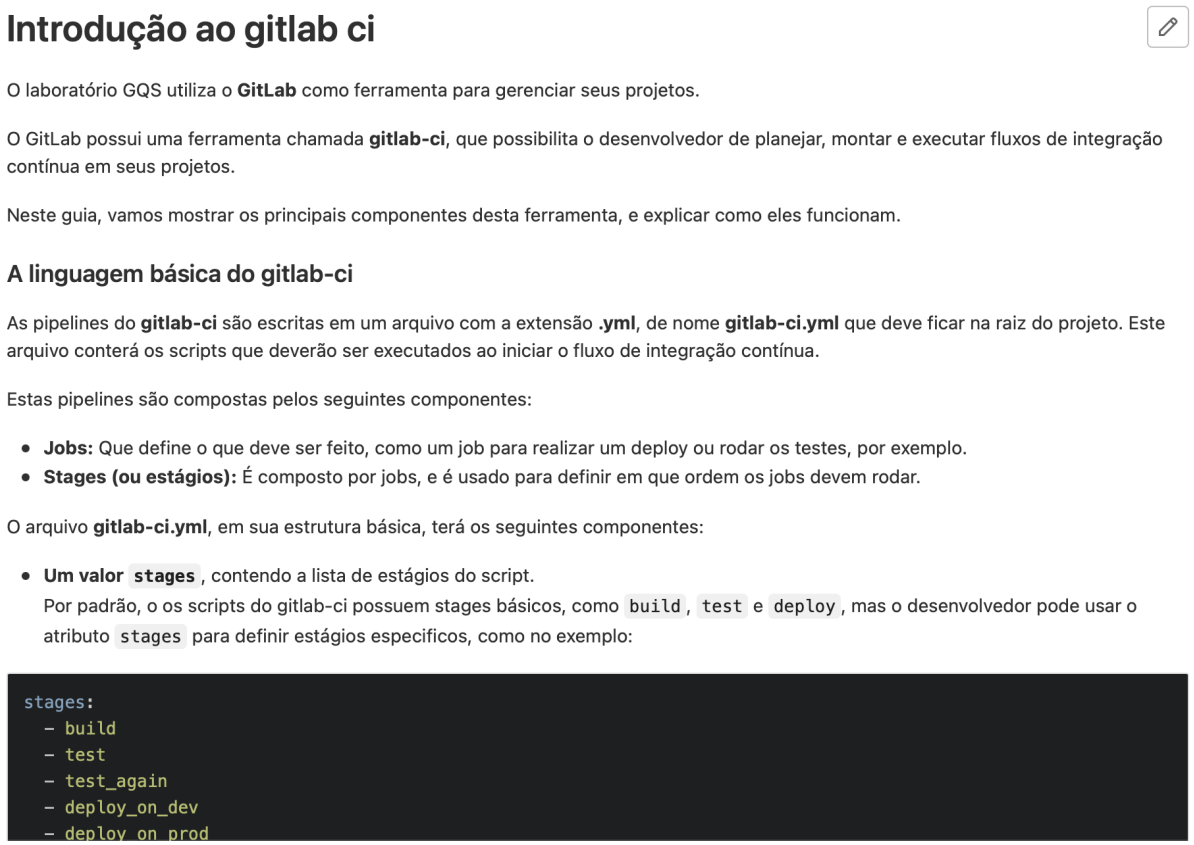
Estes conhecimentos requeridos poderiam apenas ser listados, e seria possível referenciar a documentação oficial de cada ferramenta para que os desenvolvedores pudessem pesquisar e ler mais sobre.

Entretanto, estas documentações oficiais, além de geralmente estarem em inglês, possuem muito mais informações do que o necessário para que o desenvolvedor saiba o básico para poder criar sua primeira pipeline.

Visto isso, o objetivo deste tópico é reunir toda e qualquer informação que o desenvolvedor precise saber para que possa criar sua primeira pipeline, juntando essas informações de uma maneira simples e de fácil entendimento.

4.2.1.1 GITLAB-CI

Este subtópico possui um guia introdutório que explica ao desenvolvedor como funciona a ferramenta gitlab-ci.



Introdução ao gitlab ci

O laboratório GQS utiliza o **GitLab** como ferramenta para gerenciar seus projetos.

O GitLab possui uma ferramenta chamada **gitlab-ci**, que possibilita o desenvolvedor de planejar, montar e executar fluxos de integração contínua em seus projetos.

Neste guia, vamos mostrar os principais componentes desta ferramenta, e explicar como eles funcionam.

A linguagem básica do gitlab-ci

As pipelines do **gitlab-ci** são escritas em um arquivo com a extensão **.yml**, de nome **gitlab-ci.yml** que deve ficar na raiz do projeto. Este arquivo conterá os scripts que deverão ser executados ao iniciar o fluxo de integração contínua.

Estas pipelines são compostas pelos seguintes componentes:

- **Jobs:** Que define o que deve ser feito, como um job para realizar um deploy ou rodar os testes, por exemplo.
- **Stages (ou estágios):** É composto por jobs, e é usado para definir em que ordem os jobs devem rodar.

O arquivo **gitlab-ci.yml**, em sua estrutura básica, terá os seguintes componentes:

- **Um valor** `stages`, contendo a lista de estágios do script.
  Por padrão, o os scripts do gitlab-ci possuem stages básicos, como `build`, `test` e `deploy`, mas o desenvolvedor pode usar o atributo `stages` para definir estágios especificos, como no exemplo:

```
stages:
  - build
  - test
  - test_again
  - deploy_on_dev
  - deploy_on_prod
```

Figura 27: Guia de introdução ao gitlab-ci, página inicial
(Fonte: elaborado pelo autor)

O gitlab-ci é a ferramenta principal das pipelines de CI/CD do GitLab, sendo ela o motor no qual as pipelines são construídas. Os desenvolvedores precisam ao menos entendê-la para que possam começar a escrever as suas pipelines.

Este tópico agrega todas as informações básicas sobre a ferramenta, se baseando nas informações que foram apresentadas no capítulo 2.6 deste trabalho, como por exemplo: o que são ***jobs*** e ***stages***, como as pipelines devem ser escritas, e quais os principais componentes de uma pipeline de CI/CD.

4.2.1.2 DOCKER

Este subtópico possui um guia introdutório que explica ao desenvolvedor como funciona a ferramenta Docker.

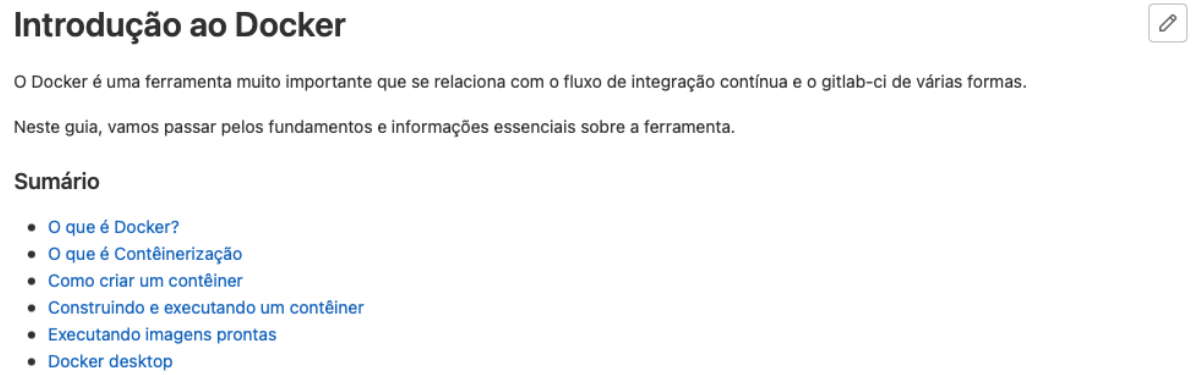


Figura 27: Guia de introdução ao Docker (introdução)
(Fonte: elaborado pelo autor)

O Docker, como dito anteriormente, é também uma ferramenta muito importante para os fluxos de CI/CD, visto que ela possui uma série de ferramentas que facilitam muito a administração de ambientes de desenvolvimento, e de execução de projetos.

Além disso, a ferramenta Docker se relaciona bastante com as pipelines de CI/CD do gitlab-ci, visto a capacidade das pipelines de utilizar imagens Docker para rodar os comandos das pipelines.

Este tópico visa explicar a fundo o que é o Docker, e introduzir de maneira simples ao desenvolvedor o conceito de conteinerização.

É mostrado também como a linguagem Docker funciona e quais seus componentes principais, possuindo uma série de exemplos práticos de como criar Dockerfiles e construir imagens para que o desenvolvedor possa entender, na prática, como esta ferramenta funciona.

4.2.1.3 RUNNERS

Este subtópico possui um guia introdutório que explica ao desenvolvedor como funcionam os Runners do gitlab-ci.

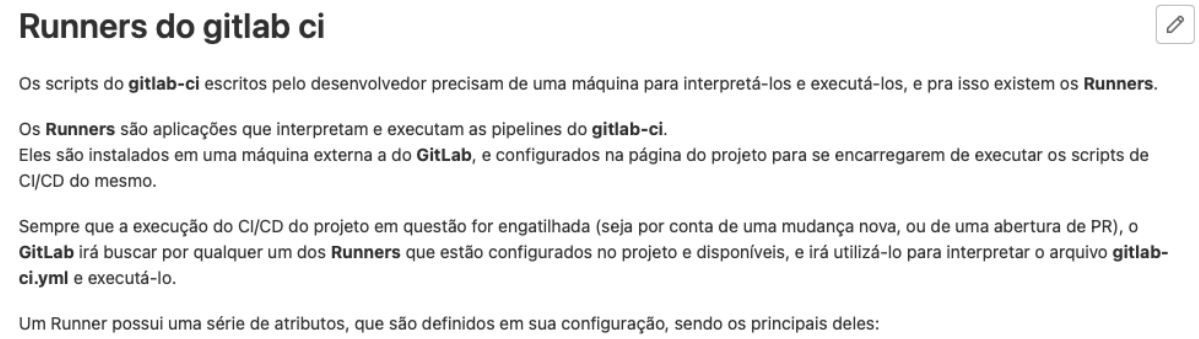


Figura 28: Guia de introdução aos Runners do GitLab, página inicial
(Fonte: elaborado pelo autor)

Como dito anteriormente, os Runners são uma parte integral e muito importante das pipelines de integração contínua. É preciso entender primeiramente o que são runners e quais suas configurações necessárias antes que o desenvolvedor possa criar a sua pipeline.

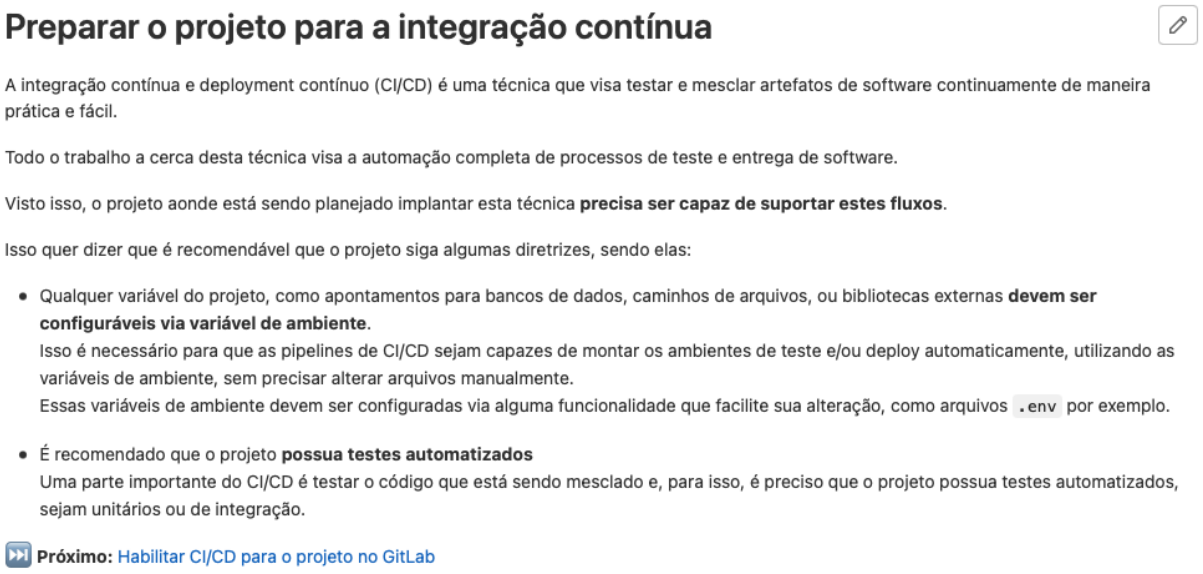
Visto isso, esse tópico se baseia no capítulo 2.6.2 deste trabalho para mostrar ao leitor como os runners funcionam, quais são seus atributos, quais são seus executores possíveis e, principalmente, qual o executor preferível no contexto do laboratório GQS.

4.2.2 TÓPICO 2: COMO CONFIGURAR A PIPELINE DE CI/CD

Com o tópico de introdução concluído, o desenvolvedor deve possuir toda a fundamentação teórica necessária para configurar a pipeline de CI/CD, e, além disso, entender o que está configurando.

Este tópico reúne todas as informações apresentadas na fundamentação teórica para indicar um passo a passo que o desenvolvedor pode seguir para ter a sua primeira pipeline de CI/CD rodando no projeto.

4.2.2.1 PREPARAR O PROJETO



Preparar o projeto para a integração contínua

A integração contínua e deployment contínuo (CI/CD) é uma técnica que visa testar e mesclar artefatos de software continuamente de maneira prática e fácil.

Todo o trabalho a cerca desta técnica visa a automação completa de processos de teste e entrega de software.

Visto isso, o projeto aonde está sendo planejado implantar esta técnica **precisa ser capaz de suportar estes fluxos**.

Isso quer dizer que é recomendável que o projeto siga algumas diretrizes, sendo elas:

- Qualquer variável do projeto, como apontamentos para bancos de dados, caminhos de arquivos, ou bibliotecas externas **devem ser configuráveis via variável de ambiente**.
  Isso é necessário para que as pipelines de CI/CD sejam capazes de montar os ambientes de teste e/ou deploy automaticamente, utilizando as variáveis de ambiente, sem precisar alterar arquivos manualmente.
  Essas variáveis de ambiente devem ser configuradas via alguma funcionalidade que facilite sua alteração, como arquivos `.env` por exemplo.

- É recomendado que o projeto **possua testes automatizados**
  Uma parte importante do CI/CD é testar o código que está sendo mesclado e, para isso, é preciso que o projeto possua testes automatizados, sejam unitários ou de integração.

⏭ **Próximo:** Habilitar CI/CD para o projeto no GitLab

Figura 29: Guia de preparação do projeto para CI/CD
(Fonte: elaborado pelo autor)

O fluxo de CI/CD, como dito anteriormente, é um fluxo que visa testar e mesclar artefatos de software continuamente e automaticamente, para manter uma qualidade e consistência na entrega de um projeto. Entretanto, para que um projeto seja capaz de implementar uma pipeline de CI/CD ideal e completa, é recomendável que ele possua algumas características chave.

Este subtópico apresenta algumas dessas características, sendo elas:

- **Possuir configuração via variável de ambiente:**
  Isto é recomendável para que a pipeline de CI/CD possa ser executada independente de quaisquer configurações manuais no projeto, como foi feito para o CodeMaster, bastando realizar configurações na própria pipeline para definir apontamentos e outras variáveis.

- **Possuir testes automatizados.**
  Isto é recomendável para que, sempre que a pipeline for executada, o código que for integrado passe por uma bateria de testes que validem regras de negócios e funcionamentos da aplicação.

4.2.2.2 HABILITAR CI/CD NO PROJETO

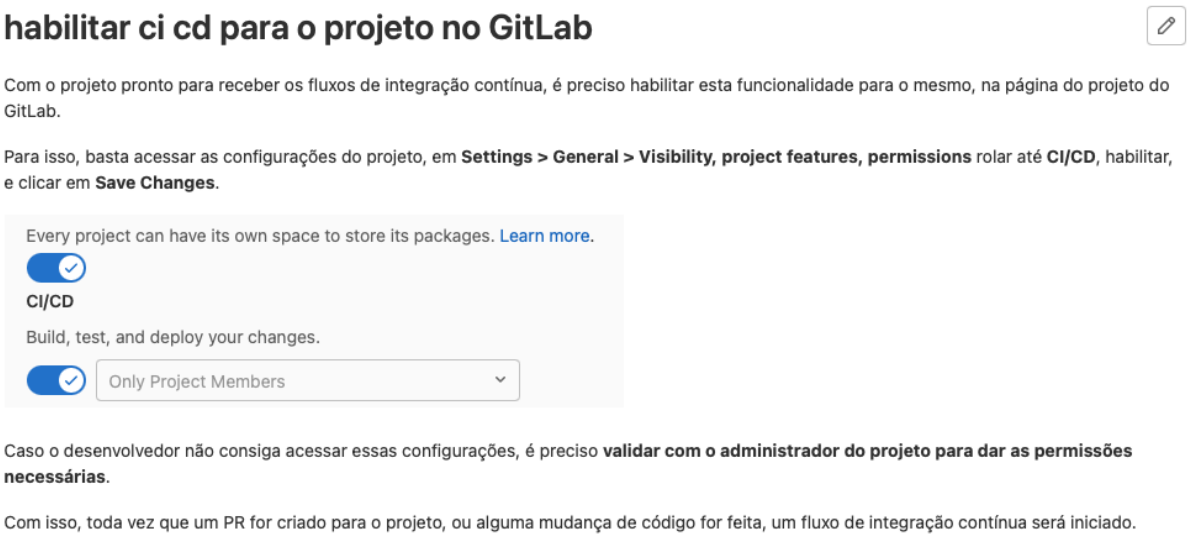


Figura 30: Guia de habilitação de CI/CD no projeto
(Fonte: elaborado pelo autor)

Para que o fluxo de CI/CD seja habilitado no projeto, é preciso primeiro realizar uma configuração simples na página do projeto no GitLab. Este breve tópico indica qual esta configuração, e onde encontrá-la.

4.2.2.3 CADASTRAR RUNNERS

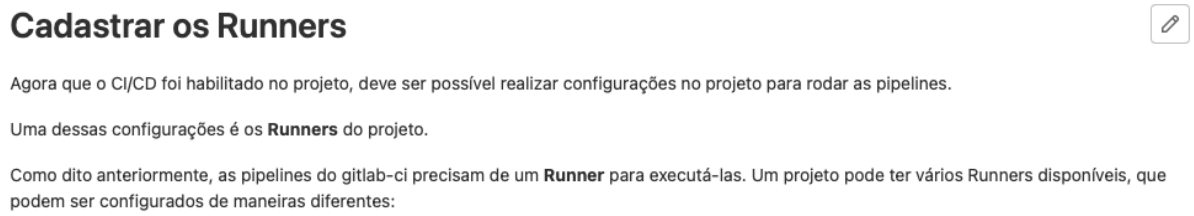


Figura 30: Guia de cadastro de Runners (introdução).
(Fonte: elaborado pelo autor)

O próximo passo para configurar a pipeline de CI/CD é configurar um Runner para o projeto. Este tópico explica ao desenvolvedor como realizar a configuração deste Runner na máquina de testes do GQS disponibilizada pela SeTIC, se baseando no que foi feito para o CodeMaster no capítulo 4.1.3.2 deste trabalho.

Este tópico também apresenta uma maneira alternativa de configurar estes Runners (Figura 31):

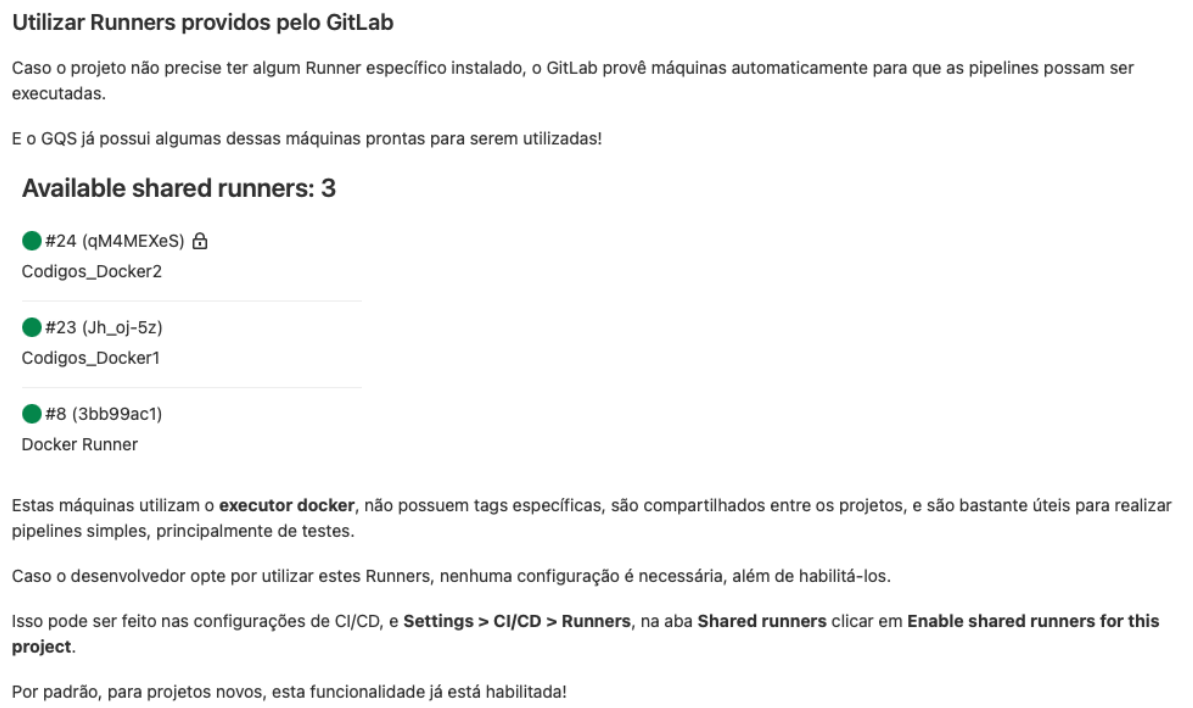


Figura 31: Guia de cadastro de Runners (runners do GitLab).
(Fonte: elaborado pelo autor)

Os Runners providos pelo GitLab são Runners que já estão disponíveis para qualquer projeto do laboratório, que podem ser utilizados pelos desenvolvedores em casos de uso mais simples.

4.2.2.4 CRIAR UMA IMAGEM BASE PARA O PROJETO

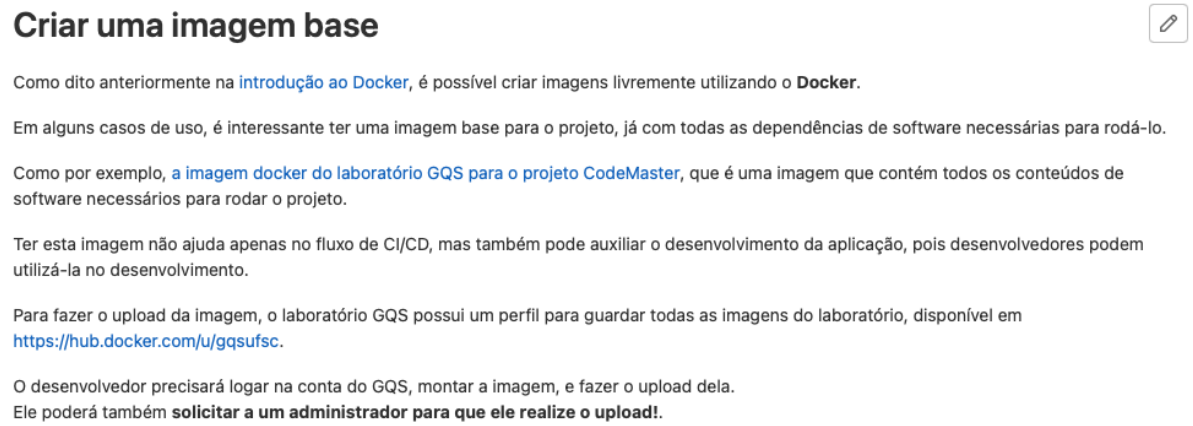


Figura 32: Guia de criação de imagens base (introdução).
(Fonte: elaborado pelo autor)

Da mesma forma que foi feito para o CodeMaster, é interessante para a maioria dos projetos a criação de uma imagem Docker base que já possua todos os requisitos de software necessários para rodar a pipeline de integração contínua.

Este tópico explica ao desenvolvedor não só como criar esta imagem Docker base, mas também como realizar o upload dela para a página do laboratório GQS no Docker.

1. ter o docker instalado na máquina
2. Clonar o projeto https://codigos.ufsc.br/gqs/ci-cd-gqs e adicionar um novo arquivo Dockerfile.
3. Escrever a lógica necessária para preparar a imagem neste Dockerfile, com base no que foi explicado na introdução ao Docker. Para este passo, o desenvolvedor poderá utilizar como base (no comando `FROM`) uma das imagens base do laboratório GQS.
   Pois o laboratório já contém uma série de imagens pré-prontas que pode ser úteis como ponto de partida para criar uma imagem nova.
4. Fazer login no docker, executando o comando `sudo docker login -p {SENHA} -u {USUARIO}`;
   Nota: Conseguir o usuário e senha do docker-hub do GQS com o administrador do laboratório!
5. Construir a imagem docker: `sudo docker build --rm -t gqsufsc/my-image:latest .`;
   Aonde `my-image` é o nome que o desenvolvedor quer dar para a imagem.
6. Fazer push da imagem: `sudo docker push gqsufsc/my-image:latest`.
7. Integrar este novo Dockerfile na branch `main` do projeto https://codigos.ufsc.br/gqs/ci-cd-gqs

Figura 33: Guia de criação de imagens base
(passo a passo de criação da imagem Docker).
(Fonte: elaborado pelo autor)

4.2.2.5 CRIAR UM DOCKERFILE PARA O PROJETO

Criar um Dockerfile para o projeto

Outra facilidade que o desenvolvedor pode implementar, é escrever um **Dockerfile** para o projeto.

Este **Dockerfile** ficará encarregado de executar a aplicação. Isto facilita muito o fluxo de CI/CD e de desenvolvimento, pois para construir e rodar a aplicação basta utilizar a ferramenta **Docker**.

Para realizar isso, basta o desenvolvedor criar um arquivo `Dockerfile` na raiz do projeto, com base no que foi indicado na Introdução ao Docker. Como imagem base para este Dockerfile (comando `FROM`) ele poderá utilizar uma das imagens bases do GQS ou a imagem base que ele criou no passo anterior!

E então a aplicação pode ser rodada com os comandos:

- `sudo docker build --rm -t my-image .`, para construir a imagem;
- `sudo docker run my-image`, para rodar a imagem.

Algo que poderá ser feito tanto em tempo de desenvolvimento, quanto na pipeline de CI/CD.

Figura 34: Guia de criação de Dockerfiles.
(Fonte: elaborado pelo autor)

Como também foi feito para o CodeMaster, é interessante para os fluxos de CI/CD que o projeto possua um Dockerfile na sua raiz que possua toda a lógica de execução do projeto, e seja capaz de executá-lo.

Este tópico explica ao desenvolvedor brevemente qual a motivação de realizar esta configuração, indicando como criar um Dockerfile, e como construí-lo em uma imagem Docker, para que seja executada em um container.

4.2.2.6 CRIAR A PRIMEIRA PIPELINE

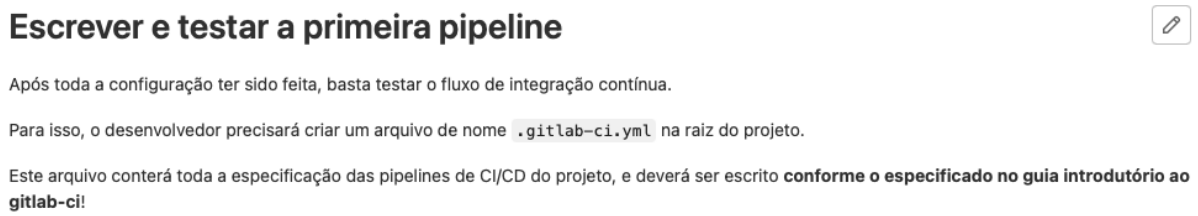


Figura 35: Guia de criação da primeira pipeline de CI/CD.
(Fonte: elaborado pelo autor)

Por fim, tendo toda a configuração pronta, este tópico indica ao desenvolvedor como criar e testar a sua primeira pipeline simples de CI/CD. Este tópico apresenta também uma pipeline de "modelo" funcional, para que os desenvolvedores possam ter um ponto de partida e possam testar as configurações feitas, sendo ela:

```
image: gqsufsc/codemaster:latest

stages:
  - build
  - test
  - deploy

build application:
  stage: build
  script:
    - echo "BUILD APPLICATION!"

test application:
  stage: test
  script:
    - echo "TEST APPLICATION!"

deploy application:
  stage: deploy
  script:
    - echo "DEPLOY APPLICATION!"
```

Esta pipeline possui três estágios básicos que executam comandos simples. Os desenvolvedores podem utilizá-la para validar se as configurações foram feitas corretamente.

4.2.3 TÓPICO 3: TÓPICOS AVANÇADOS

Por fim, como um pequeno extra, é apresentado também ao final do guia alguns tópicos extras avançados no contexto das tecnologias e ferramentas que foram apresentadas.

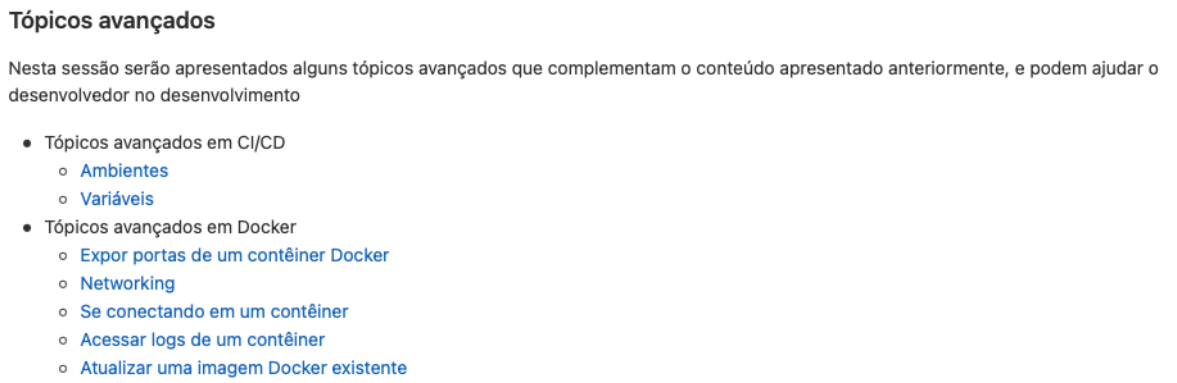


Figura 36: Tópicos avançados do guia de CI/CD do GQS.
(Fonte: elaborado pelo autor)

Estes tópicos foram criados a fim de ajudar os desenvolvedores em casos de uso mais complexos das ferramentas ***gitlab-ci*** e ***Docker***, reunindo todas as dificuldades que o autor deste trabalho passou ao criar a pipeline de CI/CD do CodeMaster, a fim de evitar que novos desenvolvedores passem pelos mesmos empecilhos.

4.3 CONSIDERAÇÕES FINAIS DA IMPLEMENTAÇÃO

Neste capítulo são apresentadas as considerações finais relacionadas a implementação do projeto. Como primeiro passo para o desenvolvimento de um Guia para definir o processo, um processo exemplo de CI/CD é definido para o projeto CodeMaster. São também desenvolvidas imagens Docker e realizadas diversas configurações, além da elaboração de um guia específico de CI/CD para a ferramenta.

Após o planejamento e implementação do processo de exemplo, um Guia geral de CI/CD para o grupo de pesquisas GQS é elaborado. Esse Guia possui as seções:

1. **O que estudar:**
   Que mostra ao desenvolvedor os conceitos teóricos básicos necessários para realizar a implementação da pipeline

2. **Como configurar a pipeline de CI/CD:**
   Que mostra ao desenvolvedor quais as configurações necessárias que ele precisa fazer para habilitar a funcionalidade de CI/CD no projeto no GitLab, e implementar a sua primeira pipeline básica.

Desta maneira, o grupo de pesquisas GQS agora possui um guia completo para que os pesquisadores do grupo consigam entender o que é CI/CD, quais seus componentes, e implementar pipelines básicas em projetos novos ou já existentes.

No próximo capítulo deste trabalho, uma avaliação do processo de CI/CD definido é apresentada.

5. AVALIAÇÃO

Este capítulo apresenta a avaliação inicial do processo de CI/CD desenvolvedor e dos guias gerados. Esta avaliação visa medir a facilidade com que os desenvolvedores absorveram as informações do guia, bem como identificar se os guias servem aos seus propósitos.

A avaliação é realizada com dois perfis de usuários do processo de CI/CD:

- Desenvolvedores que estão iniciando o trabalho com a ferramenta CodeMaster;
- Desenvolvedores que já têm prática no desenvolvimento da ferramenta.

Os dois grupos avaliam o guia de desenvolvimento criado para a plataforma CodeMaster, seguindo os passos do guia e rodando a aplicação utilizando os Dockerfiles gerados. Porém, o segundo grupo, mais experiente, também irá avaliar o funcionamento da ferramenta, verificando se todas as partes do projeto estão funcionando corretamente.

5.1 PLANEJAMENTO DA AVALIAÇÃO

A avaliação aplicada objetiva avaliar usabilidade e utilidade do processo de CI/CD para o grupo de pesquisas GQS. Os participantes desta avaliação foram convidados a participar via e-mail, e todos são pesquisadores do laboratório GQS.

Os participantes foram separados em dois perfis:

- **Pesquisadores iniciantes:**
  Pesquisadores que tiveram pouco ou nenhum contato com a ferramenta CodeMaster, e estão iniciando o desenvolvimento nela. Portanto, ainda não possuem o projeto rodando em suas máquinas pessoais.
  Estes pesquisadores possuem idade variada, e são alunos de graduação ou mestrado e doutorado.

- **Pesquisadores experientes:**
  Pesquisadores que já tiveram amplo contato com a ferramenta CodeMaster. Portanto, já possuem o projeto rodando em suas máquinas pessoais, e têm experiência o suficiente para testar as funcionalidades do projeto executado através do guia.
  Estes pesquisadores possuem idade variada, e são alunos de mestrado.

Ao separar os participantes nestes dois perfis, se objetiva identificar dois pontos principais:

- Se desenvolvedores inexperientes, que tiveram pouco ou nenhum contato com a ferramenta Docker, ou com o projeto CodeMaster, conseguem entender os conteúdos explicados no guia, e executar a aplicação em suas máquinas pessoais utilizando os Dockerfiles gerados.

- Se o ambiente de desenvolvimento executado através dos Dockerfiles é capaz de executar a ferramenta CodeMaster com sucesso, com todas as suas funcionalidades principais funcionando corretamente.

5.1.1 INSTRUMENTO DE COLETA DE DADOS

O formulário de avaliação foi criado utilizando a ferramenta google forms (GOOGLE, 2022), e disponibilizado a alguns membros do laboratório GQS para que possam respondê-lo após ter seu primeiro contato com o guia.

Existem três tipos de perguntas do formulário criado:

- **Múltipla escolha:** Campos com uma série de escolhas possíveis, e o entrevistado deverá escolher apenas uma.
- **Texto:** Campos com caixas de texto livre onde os desenvolvedores podem escrever algo.
- **Grade de múltipla escolha:** Campos com uma série de perguntas, sendo que, a cada pergunta, o entrevistado deverá escolher uma opção dentre uma escala, de "Discordo totalmente" até "Concordo totalmente".

Neste capítulo são apresentadas as perguntas do formulário, indicando também quais seus tipos. As perguntas de múltipla escolha foram separadas em positivas ou negativas, a fim de facilitar a amostragem dos resultados.

- **Pergunta 1:** Os pré-requisitos recomendados, presentes na sessão inicial do Guia, são suficientes para a utilização do Guia?
  Tipo: Múltipla escolha
  Respostas possíveis: Sim (positivo), Não (negativo)

- **Pergunta 2:** Qual das opções dadas na sessão "Como rodar o CodeMaster na máquina local" você utilizou para rodar o projeto?
  Tipo: Texto

- **Pergunta 3:** Você precisou realizar alguma configuração além das que estavam descritas na sessão "Preparar o CodeMaster para o ambiente de desenvolvimento local" para rodar o projeto?
  Tipo: Múltipla escolha
  Respostas possíveis: Sim (negativo), Não (positivo)

- **Pergunta 4:** O guia foi suficiente para rodar o projeto CodeMaster?
  Tipo: Múltipla escolha
  Respostas possíveis: Sim (positivo), Não (negativo)

- **Pergunta 5:** Como você avaliaria a usabilidade do guia?
  Esta pergunta foi feita baseada no System Usability Scale (SUS) (USABILITY.GOV, 2022), a fim de medir a usabilidade do guia utilizando

métricas oficiais reconhecidas na literatura.
Tipo: Grade de Múltipla escolha
Perguntas:
  - Eu acho o guia desnecessariamente complexo;
  - Eu achei o guia fácil de usar;
  - Eu precisei de ajuda de uma pessoa com conhecimentos técnicos para usar o guia;
  - Eu acho que as várias partes do guia estão muito bem integradas;
  - Eu acho que o guia apresenta muita inconsistência;
  - Eu precisei aprender várias coisas novas antes de conseguir usar o guia;
  - Eu imagino que as pessoas aprenderão como usar esse guia rapidamente.

- **Pergunta 6:** A ordem em que os passos do guia foram colocados é adequada?
  Tipo: Múltipla escolha
  Respostas possíveis: Sim (positivo), Não (negativo)

- **Pergunta 7:** Quais os principais pontos positivos do Guia?
  Tipo: Texto

- **Pergunta 8:** Quais os principais pontos negativos do Guia?
  Tipo: Texto

- **Pergunta 9:** Você tem alguma sugestão de melhoria para o guia?
  Tipo: Texto

Com este formulário é esperado que seja possível, além de mensurar a qualidade do guia, apontar melhorias para o mesmo, para que o trabalho feito possa alcançar uma qualidade ainda maior.

5.1.2 PLANEJAMENTO DA COLETA DE DADOS

Como dito anteriormente, as perguntas planejadas foram colocadas em um formulário de pesquisa utilizando a ferramenta Google Forms. Este formulário foi disponibilizado aos participantes da pesquisa através do seguinte link: https://forms.gle/9KnbGVSRMo8dc7Gh9.

Os participantes com o perfil iniciante foram contatados por meio de e-mail, onde foi explicado brevemente a localização e componentes do guia de desenvolvimento do CodeMaster. Ao final do email, foi enviado o link para o formulário gerado, e pedido para que o participante, após seguir o guia, respondesse às perguntas da pesquisa.

Já os participantes com o perfil experiente, foi realizada uma reunião remota utilizando a plataforma Meet, da Google, a fim de explicar mais a fundo os

componentes do guia, e mostrar também um pouco do que foi feito no projeto para implementar o processo de CI/CD.

Também, na mesma reunião, com alguns dos pesquisadores experientes, foi testado o servidor de desenvolvimento que foi executado na pipeline de CI/CD para validar se os componentes principais da aplicação estavam funcionando corretamente, a fim de ter uma avaliação preliminar do ambiente de desenvolvimento criado para o CodeMaster.

Ao final da reunião, foi pedido para que os participantes seguissem o guia de desenvolvimento, utilizando especificamente os Dockerfiles para executar a aplicação CodeMaster e, ao fim do guia, testar a aplicação, para validar que seus componentes principais estão funcionando corretamente.

5.2 RESULTADOS

Como dito anteriormente, na reunião com alguns dos desenvolvedores experientes, foi testado as principais funcionalidades do servidor de desenvolvimento que foi levantado pela pipeline de CI/CD. Nestes testes, não foi identificado nenhum problema com a aplicação, e todos os componentes principais, como os de avaliação e machine learning, estavam funcionando perfeitamente.

Quanto ao formulário de avaliação, foram entrevistados 4 pesquisadores do grupo GQS, dois com perfil **iniciante**, e dois com perfil **experiente.**

Estes foram os resultados da pesquisa:

- **Pergunta 1:** Os pré-requisitos recomendados, presentes na sessão inicial do Guia, são suficientes para a utilização do Guia?

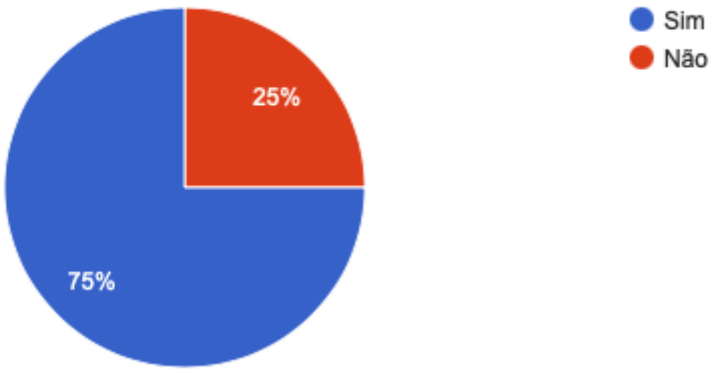


Figura 37: Resultado da pergunta 1 do formulário.
(Fonte: elaborado pelo autor)

- **Pergunta 2:** Qual das opções dadas na sessão "Como rodar o CodeMaster na máquina local" você utilizou para rodar o projeto?

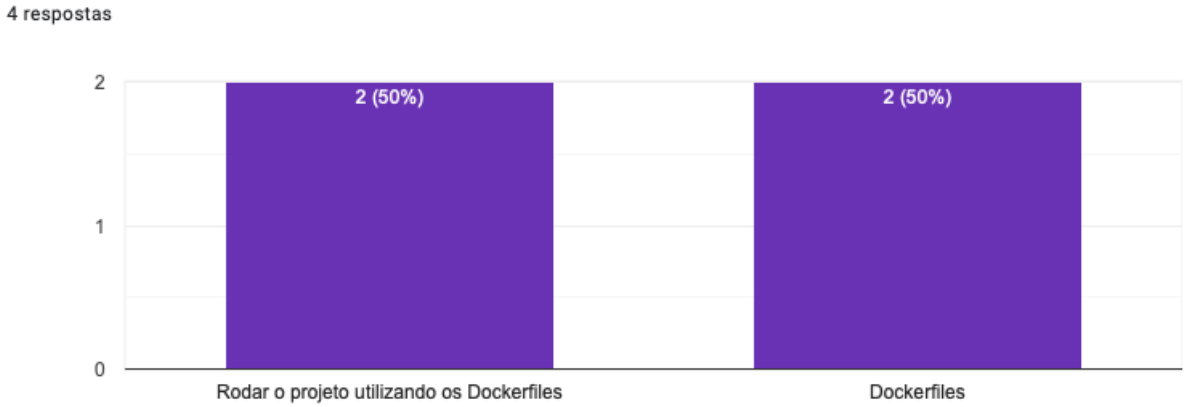


Figura 38: Resultado da pergunta 2 do formulário.
(Fonte: elaborado pelo autor)

- **Pergunta 3:** Você precisou realizar alguma configuração além das que estavam descritas na sessão "Preparar o CodeMaster para o ambiente de desenvolvimento local" para rodar o projeto?

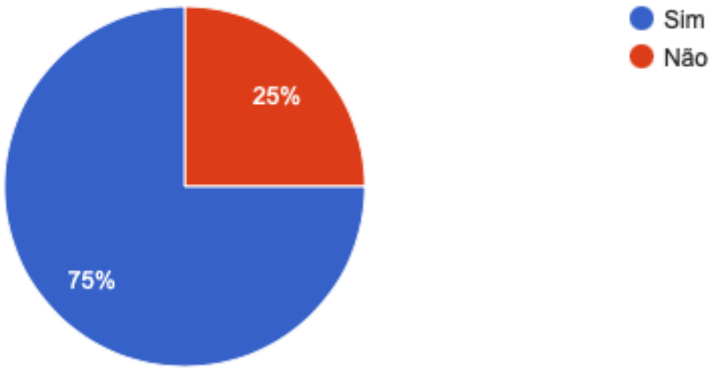


Figura 39: Resultado da pergunta 3 do formulário (parte 1).
(Fonte: elaborado pelo autor)

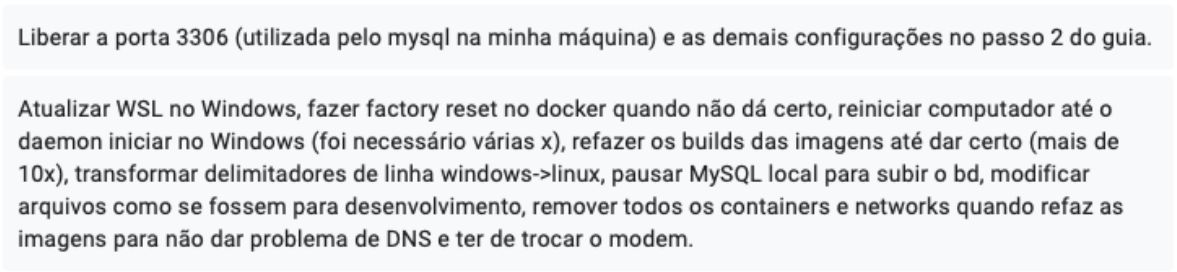


Figura 40: Resultado da pergunta 3 do formulário (parte 2).
(Fonte: elaborado pelo autor)

- **Pergunta 4:** O guia foi suficiente para rodar o projeto CodeMaster?

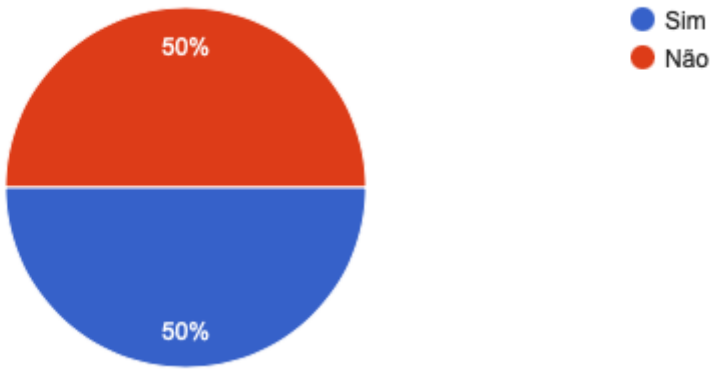


Figura 41: Resultado da pergunta 4 do formulário.
(Fonte: elaborado pelo autor)

- **Pergunta 5:** Como você avaliaria a usabilidade do guia?
  - **Eu acho o guia desnecessariamente complexo**
    - 1 → Discordo totalmente;
    - 3 → Discordo parcialmente;
  - **Eu achei o guia fácil de usar**
    - 1 → Discordo parcialmente;
    - 2 → Concordo parcialmente;
    - 1 → Concordo totalmente;
  - **Eu precisei de ajuda de uma pessoa com conhecimentos técnicos para usar o guia**
    - 1 → Discordo parcialmente;
    - 1 → Não concordo, nem discordo;
    - 1 → Concordo parcialmente;
    - 1 → Concordo totalmente;
  - **Eu acho que as várias partes do guia estão muito bem integradas**
    - 1 → Discordo totalmente;
    - 1 → Não concordo, nem discordo;
    - 2 → Concordo totalmente;
  - **Eu acho que o guia apresenta muita inconsistência**
    - 2 → Discordo totalmente;
    - 1 → Não concordo, nem discordo;
    - 1 → Concordo parcialmente;

- **Eu precisei aprender várias coisas novas antes de conseguir usar o guia**
  - 1 → Discordo totalmente;
  - 1 → Discordo parcialmente;
  - 1 → Não concordo, nem discordo;
  - 1 → Concordo parcialmente;
- **Eu imagino que as pessoas aprenderão como usar esse guia rapidamente**
  - 1 → Discordo parcialmente;
  - 2 → Concordo parcialmente;
  - 1 → Concordo totalmente;

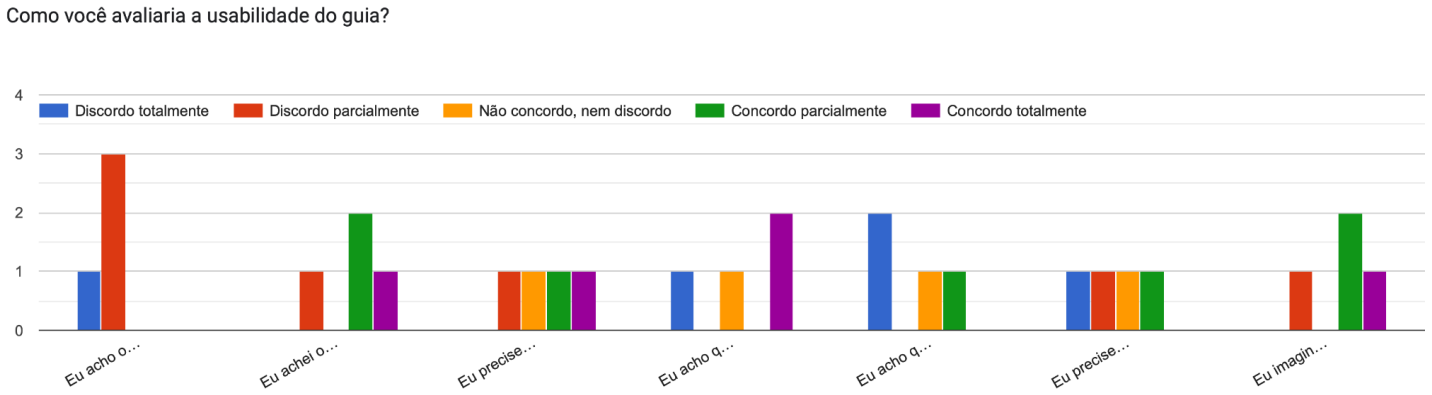


Figura 42: Resultado da pergunta 5 do formulário.
(Fonte: elaborado pelo autor)

- **Pergunta 6:** A ordem em que os passos do guia foram colocados é adequada?

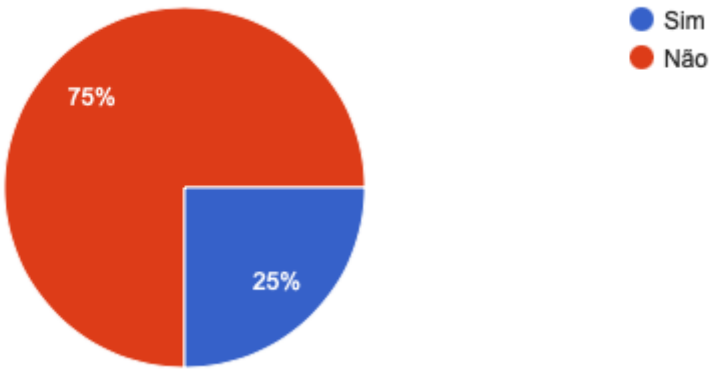


Figura 43: Resultado da pergunta 6 do formulário (parte 1).
(Fonte: elaborado pelo autor)

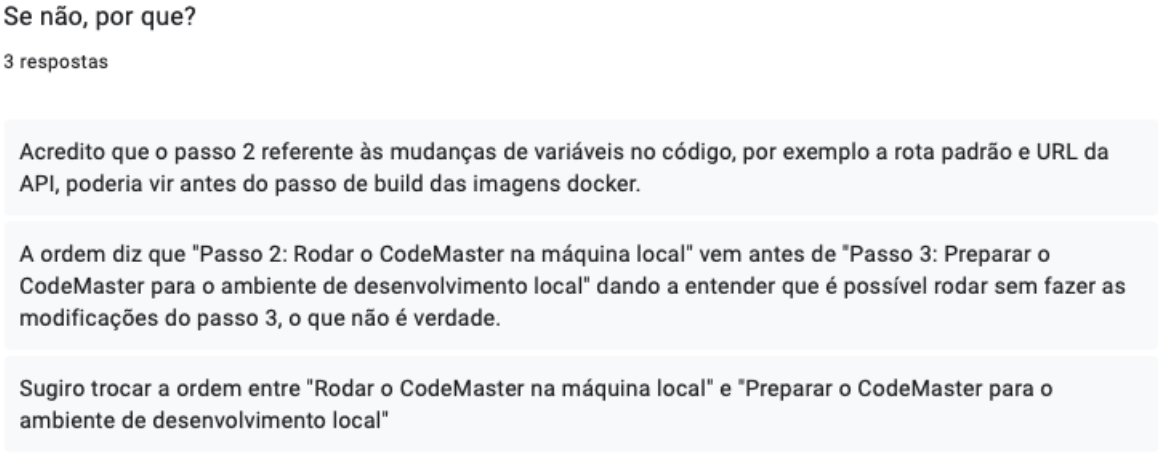


Figura 44: Resultado da pergunta 6 do formulário (parte 2).
(Fonte: elaborado pelo autor)

- **Pergunta 7:** Quais os principais pontos positivos do Guia?

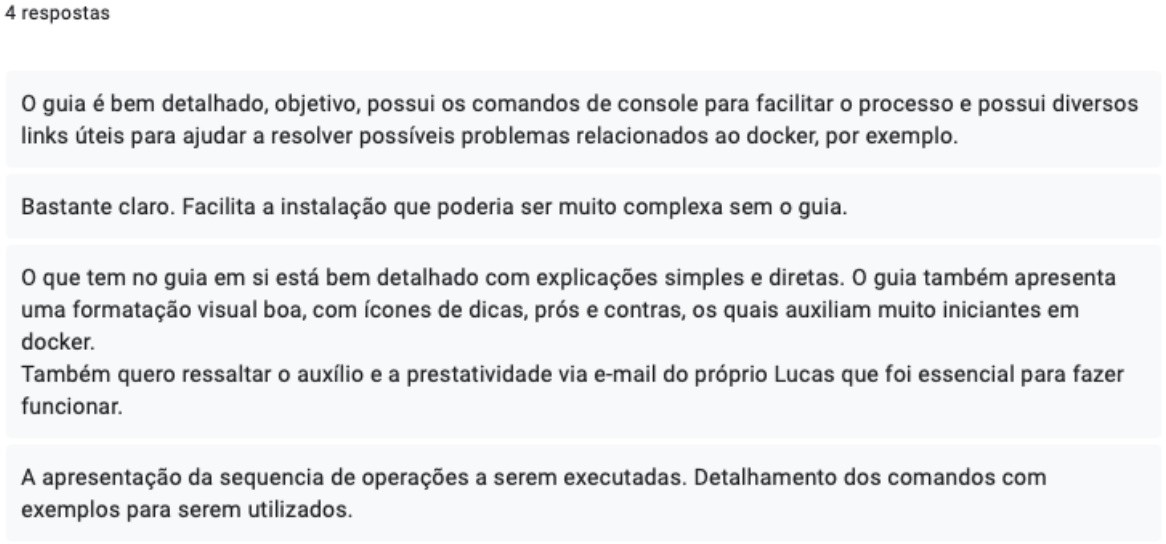


Figura 45: Resultado da pergunta 7 do formulário.
(Fonte: elaborado pelo autor)

- **Pergunta 8:** Quais os principais pontos negativos do Guia?

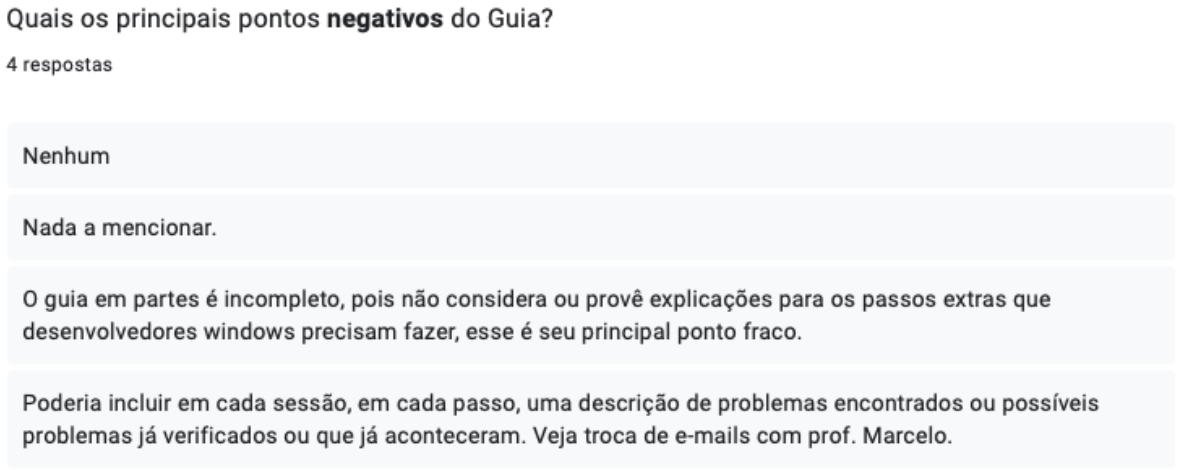


Figura 46: Resultado da pergunta 8 do formulário.
(Fonte: elaborado pelo autor)

- **Pergunta 9:** Você tem alguma sugestão de melhoria para o guia?

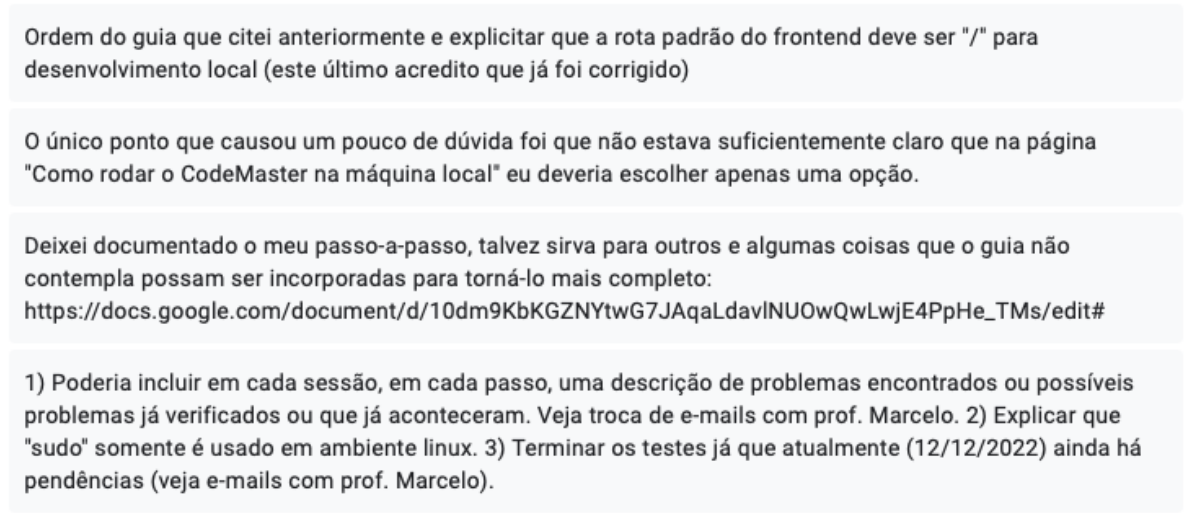


Figura 47: Resultado da pergunta 9 do formulário.
(Fonte: elaborado pelo autor)

5.2 CONSIDERAÇÕES DOS RESULTADOS DA AVALIAÇÃO

Os resultados da pesquisa de satisfação, em geral, foram positivos. Dentro do conteúdo previsto para o guia, foi possível auxiliar os pesquisadores do laboratório a configurar e rodar o projeto em suas máquinas pessoais. Contudo, alguns pontos de melhoria em relação ao guia puderam ser identificados.

O primeiro ponto de melhoria, e mais evidente, é que a ordem em que os passos do guia foram colocados não foi totalmente satisfatória para a maioria dos pesquisadores, sendo que uma boa parte deles tiveram dificuldades em seguir o guia por conta disso. Outro ponto que pôde ser identificado durante a avaliação, foi

a falta de um passo anterior ao passo 1, que indique ao desenvolvedor como baixar o projeto antes de seguir o guia.

Por conta destes dois pontos, o guia de desenvolvimento do CodeMaster, que antes possuía dois passos, foi re-estruturado em guia de quatro passos, como é possível ver na figura 48.

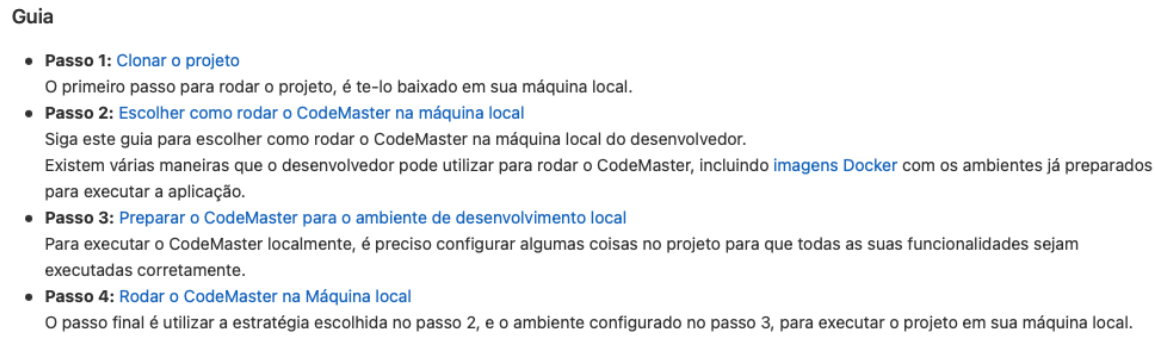


Figura 48: Guia de desenvolvimento do CodeMaster atualizado.
(Fonte: elaborado pelo autor)

Nesta nova estrutura, o guia agora é composto por quatro passos:

1. **Clonar o projeto:**
   Explica ao desenvolvedor como baixar o projeto do GitLab para sua máquina local e preparar o projeto para a utilização do guia.

2. **Escolher como rodar o CodeMaster na máquina local**
   Lista as três maneiras possíveis que o desenvolvedor pode utilizar para rodar o CodeMaster em sua máquina pessoal, listando os prós e contras de cada opção, para que o desenvolvedor possa escolher como prefere executar o projeto.

3. **Preparar o CodeMaster para o ambiente de desenvolvimento local**
   Este passo continua sendo o mesmo do passo 2 do guia original, porém desta vez contendo dicas de configurações com base na opção que o desenvolvedor escolheu no passo 2 do guia reestruturado.

4. **Rodar o CodeMaster na máquina local**
   Indica ao desenvolvedor qual o processo para utilizar a opção escolhida e o ambiente configurado para executar o projeto em sua máquina pessoal.

Outro ponto bastante evidente, que pode ser identificado na figura 41, é que o guia não foi suficiente para executar o projeto para a metade dos pesquisadores que o seguiram. Ao analisar os processos dos pesquisadores, pôde-se identificar que, em boa parte, isso ocorreu pela dificuldade da utilização do guia por desenvolvedores que utilizam máquinas Windows, principalmente por particularidades relacionadas ao Docker.

Visto isso, o passo 4 do guia foi incrementado com ações extras que os pesquisadores que utilizam Windows precisaram fazer durante o processo de utilização do guia.

A seção "Links Úteis" do guia foi também incrementada, não só com mais informações relacionadas a utilização do Docker no Windows, mas também com vários processos de solução de problemas ou funcionalidades úteis que os pesquisadores utilizaram para a avaliação do ambiente de desenvolvimento, que podem vir a ser úteis a outros pesquisadores no futuro.

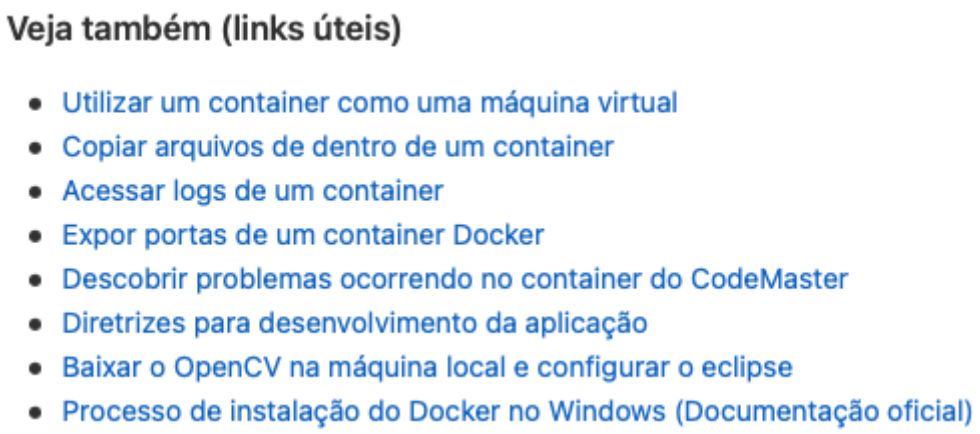


Figura 49: Seção "links úteis" do guia do CodeMaster atualizada.
(Fonte: elaborado pelo autor)

Estes resultados da avaliação, e os testes no ambiente de desenvolvimento feitos com os desenvolvedores experientes, indicam que o ambiente de desenvolvimento do projeto CodeMaster (composto pelos Dockerfiles, pipelines de CI/CD, etc) foi criado com sucesso, e aplicação agora é capaz de executar facilmente um ambiente de desenvolvimento remoto e local, com a ajuda guias explicativos que auxiliam os pesquisadores do laboratório.

6. CONCLUSÃO

Neste capítulo são apresentadas as conclusões deste trabalho. Este trabalho apresenta o desenvolvimento, documentação e implantação de um processo de Integração e *Deployment* Contínuos no grupo de pesquisas GQS.

Segundo o que foi mostrado, o objetivo principal do trabalho foi alcançado. O processo de integração e *Deployment* Contínuos foi implementado no laboratório GQS, no projeto CodeMaster, que, no presente momento, possui uma pipeline de CI/CD executada com sucesso, e um servidor de desenvolvimento no ar, que foi levantado inteiramente por esta pipeline.

Todo o processo de implementação e configuração destas pipelines na ferramenta GitLab foi inteiramente documentado em um ambiente de fácil acesso a todos os desenvolvedores do laboratório. Foi também documentada toda a fundamentação teórica necessária para implementar estas pipelines, para que desenvolvedores leigos consigam facilmente configurar e implantar novas pipelines em seus projetos.

Desta maneira, o laboratório GQS agora possui um guia teórico completo para a implementação de pipelines de CI/CD nos projetos do laboratório, e um fluxo de CI/CD completo e funcional, para servir de exemplo.

Quanto aos objetivos específicos:

- **Analisar a literatura e o estado da arte em relação a integração deployment contínuos:**
  No capítulo 2 deste trabalho, foi reunida e documentada todo o estado da arte no que diz respeito às tecnologias e conceitos relacionados à integração contínua presentes na literatura.
  Todos estes conceitos foram analisados e levados em consideração na criação do guia e da pipeline de CI/CD.

- **Analisar os sistemas desenvolvidos e a infra estrutura do GQS:**
  Para realizar a escolha do projeto onde seria implantado a pipeline de CI/CD de exemplo, foi analisada primeiramente a estrutura geral dos projetos presentes atualmente no laboratório GQS.
  O projeto CodeMaster, como dito anteriormente, foi escolhido por ser um dos maiores e mais complexos projetos no contexto do laboratório e, após ser escolhido, foi analisada a fundo toda a estrutura deste projeto, para que a pipeline de CI/CD fosse implementada da melhor maneira possível. Toda essa análise foi então documentada no capítulo 3.2.1 deste trabalho.
  A análise feita da estrutura geral dos projetos do GQS foi também utilizada para a criação das imagens base do laboratório GQS, algo que foi mostrado no capítulo 4.1.2 deste trabalho.
  Desta maneira, estas imagens base puderam ser criadas com o fim de atender os objetivos gerais dos projetos do laboratório GQS.

- **Desenvolver o processo de CI/CD:**
  Conforme mostrado no capítulo 4.1 deste trabalho, o fluxo de CI/CD foi planejado, desenvolvido e implantado com sucesso no projeto CodeMaster, que agora possui uma pipeline de CI/CD que é capaz de testar o funcionamento do código e realizar um deploy para o servidor de desenvolvimento de uma maneira automática.

- **Documentar o processo de CI/CD:**
  Como mostrado no capítulo 4.2 deste trabalho, o guia de CI/CD foi devidamente criado e disponibilizado aos desenvolvedores do laboratório GQS.
  Este guia possui toda a fundamentação teórica e prática que o desenvolvedor precisa para criar e implantar a sua primeira pipeline de CI/CD, usando a ferramenta gitlab-ci.
  Este guia foi criado utilizando a própria ferramenta GitLab do laboratório GQS, a fim de manter estas informações com um fácil acesso de leitura e escrita a todos os membros do laboratório.
  Também, como mostrado no capítulo 4.1, foram também documentadas boas práticas e novos processos no desenvolvimento e configuração do projeto CodeMaster.
  Estes processos e boas práticas que foram implantados e documentados no projeto estão alinhados com os conceitos de integração contínua, visto que facilitam o desenvolvimento da aplicação, e são conceitos e tecnologias utilizados nas pipelines de CI/CD.

- **Avaliar o processo de CI/CD:**
  Foi implantada uma avaliação neste trabalho que objetivou avaliar o guia de desenvolvimento e boas práticas criado para o CodeMaster, visto que este guia engloba alguns dos conceitos de integração contínua, e alguns dos conceitos teóricos e práticos utilizados no guia de CI/CD.
  Nos resultados desta avaliação, pôde-se identificar que o ambiente de desenvolvimento do projeto CodeMaster foi implementado com sucesso. E o guia criado para o projeto foi capaz de auxiliar os pesquisadores do laboratório no entendimento deste novo ambiente.
  Todos os pontos e sugestões de melhoria para o guia levantados pelos pesquisadores foram levados em consideração, e o guia foi incrementado.
  Quanto ao processo de CI/CD em si, pode-se ver no capítulo 4.1 que ele foi executado com sucesso, e todos os objetivos previstos para ele foram cumpridos.

Desta maneira, pode-se observar que todos os objetivos deste projeto foram devidamente alcançados, e agora foi implantando, de maneira prática e teórica, o conceito de integração e *deployment* contínuos no laboratório GQS, que agora possui um guia completo para a implementação de pipelines de CI/CD, bem como uma pipeline funcional de exemplo.

6.1 TRABALHOS FUTUROS

Dentro do contexto deste trabalho, todos os objetivos previstos foram cumpridos. Porém, existem uma série de melhorias que não foram previstas neste projeto, que poderão ser implementadas no futuro, a fim de melhorar a qualidade destes processos implementados.

A pipeline de CI/CD do CodeMaster, não possui um fluxo que realize um deploy em produção. Dentro do tempo previsto para este trabalho, isso não pôde ser realizado, porém, no futuro, essa funcionalidade poderia ser implementada no projeto, adicionando este fluxo à pipeline já existente.

Neste mesmo contexto, a pipeline de CI/CD do CodeMaster também não possui um fluxo que realize testes automatizados no projeto, algo que é altamente recomendado nos fluxos de integração contínua, a fim de testar o software antes que ele seja integrado ao ambiente de desenvolvimento ou produção. Como o projeto CodeMaster não possuía testes automatizados, isso não pôde ser feito dentro do escopo deste trabalho.

Ainda no contexto da pipeline do CodeMaster, ao subir a pipeline, é necessário realizar algumas configurações manuais no projeto para que o servidor de desenvolvimento seja executado. Isso foi feito desta maneira pois o CodeMaster não possui uma funcionalidade que possibilite a configuração remota do projeto, como por exemplo, a utilização de variáveis de ambiente que possam ser configuradas ao executá-lo, algo que melhoraria o processo de execução da pipeline implantada.

Visto isso, como trabalhos futuros, seria possível realizar uma grande melhoria na pipeline de CI/CD implementada para o CodeMaster, implantando um fluxo que realize um deploy em produção, implementando testes automatizados para o projeto, e possibilitando a configuração de todo o ambiente de desenvolvimento diretamente da pipeline, dispensando as configurações manuais que são necessárias atualmente.

Como dito anteriormente, as pipelines do CodeMaster foram criadas primeiramente em um projeto de testes, e então, ao mover a pipeline para o projeto real, o autor seguiu fielmente os passos do guia criado, a fim de validar se os passos do guia eram suficientes para configurar o CI/CD no projeto.

Porém, para uma validação profunda e mais fiel, seria necessária a disponibilidade de um ou mais desenvolvedores leigos do laboratório, para que eles pudessem ler o guia geral, e segui-lo do começo ao fim para implantar uma pipeline de CI/CD em um projeto do grupo GQS, sendo ele novo ou já existente.

Essa avaliação, por mais que não tenha sido possível fazer neste trabalho, é bastante importante para validar a qualidade do guia, bem como medir a facilidade que os desenvolvedores tiveram para absorver os conteúdos apresentados, e pode ser realizada em trabalhos futuros.

Ao implantar o conceito de CI/CD no grupo GQS, com guias e pipelines de exemplo, espera-se que sejam adotadas boas práticas no desenvolvimento de software no grupo, e a qualidade geral da entrega dos projetos do grupo aumente. Visto isso, como trabalho futuro, seria possível fazer uma avaliação a longo prazo dos efeitos que estes processos e boas práticas de CI/CD trouxeram no grupo efetivamente.

6.2 CONSIDERAÇÕES FINAIS

Como é possível observar nos resultados deste projeto, implantar o conceito de integração e deployment contínuos em um time de software pode trazer benefícios além dos que foram previamente identificados na fundamentação teórica deste trabalho.

Claramente, as pipelines de CI/CD ajudam muito no desenvolvimento de um produto de software ao automatizar todo o processo de teste e entrega do mesmo. Processo este que pode vir a ser difícil de fazer, à medida que o time, e o produto de software, crescem.

Porém, pode-se identificar outros benefícios. Ao implantar o conceito de CI/CD, o time responsável pelo produto de software acaba automaticamente adotando boas práticas que facilitam, ou até possibilitam, a implantação das pipelines de integração contínua. Como, por exemplo, o uso do Docker, uso de variáveis de ambiente e configurações remotas, a adoção de testes automatizados, entre outros.

Todas estas boas práticas que estão dentro do conceito de CI/CD melhoram significativamente a qualidade de um produto de software, bem como a facilidade a qual ele é configurado, desenvolvido, e distribuído.

Isso pôde ser identificado na implantação do fluxo de CI/CD no projeto CodeMaster, visto que era um projeto complexo de difícil manutenção, onde desenvolvedores gastavam uma grande quantidade de tempo para configurá-lo antes de conseguir trabalhar no mesmo. Agora, ao implantar a integração contínua, o projeto possui uma configuração muito mais fácil e amigável, utilizando os mesmos Dockerfiles que a pipeline de CI/CD utiliza.

Por mais que o CodeMaster tenha sido o único projeto onde a pipeline de CI/CD foi criada, foi criado também imagens Docker base para outros contextos de outros projetos do laboratório, bem como guias para a implementação de fluxos de CI/CD na plataforma GitLab, para que outros desenvolvedores possam facilmente implantar também estes conceitos e boas práticas em outros projetos do laboratório no futuro.

Ao avaliar o guia e o processo de CI/CD criado para o projeto de exemplo, pôde-se identificar que o objetivo principal do projeto foi bem sucedido. A ferramenta CodeMaster agora possui um processo de CI/CD funcional, capaz de construir qualquer novo código integrado ao ambiente de desenvolvimento e executar um servidor de testes, de maneira automática. E possui, também, um guia completo para que desenvolvedores novos ou desenvolvedores experientes possam entender e usufruir deste novo processo de integração contínua e ambiente de desenvolvimento do projeto, a fim de facilitar o processo de desenvolvimento da aplicação.

Com isso, pode ser identificada a importância deste projeto. A UFSC e o GQS possuem projetos de grande importância no âmbito social e, manter a qualidade da entrega destes projetos é algo de extrema importância.

Ao implantar o conceito de integração e *deployment* contínuos no GQS, espera-se que seja também implantada uma nova maneira de planejar, desenvolver, e entregar software no laboratório, com melhores práticas de desenvolvimento, maior qualidade nas entregas e uma maior facilidade no processo de entrega destas aplicações.

7. REFERÊNCIAS

BARIK, T et. al. **Commit bubbles**. 2015. Disponível em: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7203030

DOCKER INC. **Docker Hub,** 2021b. Hub de imagens do docker. Disponível em: <https://hub.docker.com>. Acesso em 29 de junho de 2022.

DOCKER INC. **Docker Hub,** 2022c. Imagem Docker do mysql. Disponível em: <https://hub.docker.com/_/mysql>. Acesso em 07 de novembro de 2022.

DOCKER INC. **Docker Hub,** 2022d. Imagem Docker de frontend do GQS. Disponível em: <https://hub.docker.com/r/gqsufsc/frontend>. Acesso em 07 de novembro de 2022.

DOCKER INC. **Docker Hub,** 2022e. Imagem base do projeto CodeMaster. Disponível em: <https://hub.docker.com/r/gqsufsc/codemaster>. Acesso em 07 de novembro de 2022.

DOCKER INC. **Networking Overview,** 2022f. Documentação oficial das redes Docker. Disponível em: <https://docs.docker.com/network/>. Acesso em 08 de novembro de 2022.

DOCKER INC. **Docker Overview,** 2021a. Documentação oficial do Docker. Disponível em: <https://docs.docker.com/get-started/overview/>. Acesso em 29 de junho de 2022.

EBERMANN, A. **Evaluation of GitOps Security in a CI/CD Environment**. jul. 2019. Disponível em: https://www.iteratec.com/fileadmin/assets/downloads/bachelorarbeit_gitops_security.pdf.

GÓIS, F. et. al. Desenvolvimento de Software Utilizando Integração Contínua. **ERCEMAPI 2007**, Ceará, p. 62-80, jul. 2007. Disponível em: https://www.researchgate.net/profile/Pedro-Muniz-Farias/publication/353273690_ERCEMAPI_2007_versao_final_completa_com_isbn/links/60f0890c16f9f3130087493b/ERCEMAPI-2007-versao-final-completa-com-isbn.pdf#page=62.

GIT. **Git**, 2022. Página inicial. Disponível em <https://git-scm.com/site>. Acesso em 28 de junho de 2022.

GITLAB B. V. **GitLab**, 2022a. Página inicial. Disponível em <https://about.gitlab.com>. Acesso em 28 de junho de 2022.

GITLAB B. V. **What is a Merge Request**, 2022b. Documentação oficial do GitLab. Disponível em <https://docs.gitlab.com/ee/user/project/merge_requests/>. Acesso em 28 de junho de 2022.

GITLAB B. V. **GitLab CI/CD**, 2022c. Documentação oficial do GitLab sobre CI/CD. Disponível em <https://docs.gitlab.com/ee/ci/>. Acesso em 28 de junho de 2022.

GITLAB B. V. **CI/CD pipelines**, 2022d. Documentação oficial do GitLab referente às pipelines de integração e deployment contínuos. Disponível em: <https://docs.gitlab.com/ee/ci/pipelines/index.html>. Acesso em 28 de junho de 2022.

GITLAB B. V. **Install GitLab Runner**, 2022e. Documentação oficial do GitLab referente a instalação de runners. Disponível em: <https://docs.gitlab.com/ee/ci/pipelines/index.html>. Acesso em 28 de junho de 2022.

GITLAB B. V. **Environments and deployments**, 2022f. Documentação oficial do GitLab referente a ambientes. Disponível em: <https://docs.gitlab.com/ee/ci/environments/>. Acesso em 28 de junho de 2022.

GITLAB B. V. **Wiki**, 2022g. Documentação oficial do GitLab referente a ferramenta wiki. Disponível em: <https://docs.gitlab.com/ee/user/project/wiki/>. Acesso em 24 de novembro de 2022.

GOOGLE. **Formulários Google**, 2022. Página inicial da ferramenta de formulários da google. Disponível em: <https://workspace.google.com/intl/pt-BR/products/forms/>. Acesso em 16 de novembro de 2022

GRUPO DE QUALIDADE DE SOFTWARE. **GQS - Software Quality Group**, 2022a. Página inicial. Disponível em: <http://www.gqs.ufsc.br>. Acesso em 28 de junho de 2022.

GRUPO DE QUALIDADE DE SOFTWARE. **Computação na Escola**, 2022b. Página inicial. Disponível em: <https://computacaonaescola.ufsc.br/>. Acesso em 28 de junho de 2022.

GRUPO DE QUALIDADE DE SOFTWARE. **Codemaster,** 2021. Página inicial do projeto CodeMaster. Disponível em: <http://apps.computacaonaescola.ufsc.br/codemaster/>. Acesso em 29 de junho de 2022.

IEEE STANDARDS ASSOCIATION. **IEE 2675-2021:** ***IEEE Standard for DevOps: Building Reliable and Secure Systems Including Application Build, Package, and Deployment***. Fev. 2021. Disponível em: https://standards.ieee.org/ieee/2675/6830/.

LE YEOH, J. **Different environments in a software development team.** 2021. Disponível em: https://medium.com/@Jia_Le_Yeoh/what-are-the-different-environments-in-a-software-development-team-3c94381bb496

Massachusetts Institute of Technology. **MIT App Inventor**, 2022. Página inicial. Disponível em: <https://appinventor.mit.edu>. Acesso em 20 de julho de 2022.

PINTO, A; TERRA, R. **Processo de Conformidade Arquitetural em Integração Contínua.** 2015. Disponível em: http://professores.dcc.ufla.br/~terra/publications_files/students/2015_ufla_pinto.pdf.

SCHIESTL, B. **Code Branching Definition | What is a Branch?**. fev. 2020. Disponível em: <https://www.perforce.com/blog/vcs/branching-definition-what-branch>. Acesso em 29 de junho de 2022.

TERRA, R; TÚLIO, M. **Definição de Padrões Arquiteturais e seu Impacto em Atividades de Manutenção de Software.** 2010. Disponível em: https://www.researchgate.net/publication/262563350_Definicao_de_Padroes_Arquiteturais_e_seu_Impacto_em_Atividades_de_Manutencao_de_Software.

Tom Donohue. **How To Communicate Between Docker Containers**, 2022. Disponível em: <https://www.tutorialworks.com/container-networking/>. Acesso em 8 de novembro de 2022.

Tom Donohue. **How To Communicate Between Docker Containers**, 2022. Disponível em: <https://www.tutorialworks.com/container-networking/>. Acesso em 8 de novembro de 2022.

UFSC. **SeTIC**, 2022. Página inicial. Disponível em <https://setic.ufsc.br>. Acesso em 28 de junho de 2022.

USABILITY.GOV. **System Usability Scale (SUS)**, 2022. Página inicial do sistema de escala de usabilidade. Disponível em: <https://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html>. Acesso em 16 de novembro de 2022

APÊNDICE A - ARTIGO: Implantação de um Processo de Integração e *Deployment* Contínuos em um Laboratório de Pesquisa

Lucas E. Andrade

Universidade Federal de Santa Catarina

lucas.andrade.l.a@grad.ufsc.br

Resumo. *Manter a qualidade da entrega de softwares desenvolvidos é uma tarefa complicada, que tende a aumentar em complexidade conforme o software cresce. Fluxos de Integração Contínua (ou Continuous Integration - CI) e de Deployment Contínuo (ou Continuous Deployment - CD) automatizam esses processos de entrega, integração, e até de testes de software. Assim, este projeto objetiva analisar as características dos softwares desenvolvidos no Grupo de Qualidade de Software (GQS), e definir e implantar um processo de integração contínua.*

Abstract. *Maintaining the quality of the developed software is a complicated task, which tends to increase in complexity as the software grows. Continuous Integration (or CI) and Continuous Deployment (or CD) flows automate these delivery, integration, and even software testing processes. Thus, this project aims to analyze the characteristics of the software developed in the "Grupo de Qualidade de Software" (GQS), and define and implement a continuous integration process.*

1. Introdução

Ao começar a desenvolver um software, tipicamente a arquitetura do mesmo é definida. A arquitetura de software consiste em um conjunto de decisões que um time toma sobre um projeto de software, que impactarão diretamente a maneira como o mesmo deve ser construído e mantido [Terra, Túlio 2010].

A arquitetura definida para o software começa, então, a ser implementada, e testes unitários ou de integração, verificações de qualidade de escrita de código, validações de contratos entre API's, *deploy* (implantação) do projeto, auxiliam um time a manter a integridade do software [Góis et. al. 2007].

No entanto, segundo, na medida que um software cresce, bem como o time que o desenvolve, é quase que natural que o time acabe se desviando das boas práticas de engenharia de software [TERRA 2015]. Testes vão sendo deixados para trás, padrões de arquitetura param de ser seguidos e problemas passam a ocorrer em ambientes de produção. Isso pode ocorrer por uma série de motivos, como prazos curtos, rotatividade do time, pressão dos stakeholders, etc.

Preservar padrões e boas práticas de arquitetura de um software contribui fortemente para manter a escalabilidade, manutenibilidade, reuso e portabilidade de um software [Terra, Túlio 2010]. Mas manter esses padrões definidos pelo time manualmente é uma tarefa difícil. E é aí que entram os fluxos de integração e deployment contínuos.

0 conceito de Integração Contínua, ou CI, pode ser definida como: "Técnica que continuamente mescla artefatos, incluindo atualizações de código-fonte de todos os desenvolvedores de uma equipe, em uma linha principal compartilhada para construir e testar o sistema desenvolvido."[4] . [IEE 2675 2021]

Já *deployment* contínuo, ou CD, é o "Processo automatizado de implantação de alterações na produção, realizando validações a fim de reduzir riscos."[5]. [IEE 2675 2021]

Ou seja, em outras palavras, os fluxos de CI e CD, são fluxos automatizados para realizar alguma rotina em um projeto. Eles facilitam muito a vida do desenvolvedor ou de um time de desenvolvedores, principalmente quando se trata de um projeto grande. Com um fluxo de CI bem estabelecido, um desenvolvedor consegue definir rotinas automáticas que garantam um bom funcionamento de um projeto, ou até que realizem a entrega do mesmo.

O GQS é um grupo de pesquisas do Departamento de Informática e Estatística da Universidade Federal de Santa Catarina. Os pesquisadores do GQS desenvolvem pesquisas na área de engenharia de software, com a finalidade de identificar e elaborar processos para melhorias na qualidade e produtividade no desenvolvimento de softwares [GRUPO DE QUALIDADE DE SOFTWARE, 2022]. O laboratório também possui programas de ensino na área de computação em escolas.

Os projetos implementados no GQS são gerenciados na plataforma GitLab [GITLAB B. V 2022a], que é hospedada em um servidor da UFSC. A plataforma GitLab possui um grande suporte para fluxos de integração contínua e deployment contínuo (CI/CD), possuindo

[4] tradução própria, original: "*Technique that continually merges artifacts, including source code updates from all developers on a team, into a shared mainline to build and test the developed system.*

[5] "tradução própria, original: "*Automated process of deploying changes to production by verifying intended features and validations to reduce risk.*"

uma documentação extensa que facilita bastante para o desenvolvedor implementar essa funcionalidade em novos projetos ou projetos já existentes.

No entanto, apesar dos projetos desenvolvidos no GQS possuírem alto impacto na sociedade, e a plataforma em que os projetos se encontram possua suporte para CI/CD, a grande maioria deles não possui fluxos de CI/CD. As entregas são feitas manualmente, bem como as verificações de testes, sem falar que alguns dos projetos são desenvolvidos por mais de uma pessoa, e ter um fluxo de *linting* nestes casos seria interessante para manter um estilo e qualidade de escrita de código constante.

Espera-se que o desenvolvimento, implantação e documentação de um processo de integração e Deployment contínuos no grupo de pesquisas GQS, possibilite um fluxo de entrega e qualidade constante de fácil uso.

2. Proposta

A proposta principal deste projeto consiste no desenvolvimento e documentação de um processo de CI/CD para os projetos do grupo de pesquisas GQS. O processo é documentado no formato de um guia, para que os pesquisadores do grupo consigam implementar estes fluxos nos projetos novos ou já existentes.

Para a confecção deste guia, é realizada a implementação do fluxo de CI/CD em um projeto de exemplo, para que sirva como base teórica e prática da documentação. Visto isso, este capítulo inicia com definição da estrutura do guia de CI/CD que será desenvolvido e em seguida é apresentada, como exemplo, a implementação de um processo de CI/CD em um dos projetos já existentes do laboratório GQS.

Dentre os projetos que estão inseridos dentro do contexto do GQS, este capítulo apresenta como exemplo a definição de um processo de CI/CD no projeto CodeMaster [GRUPO DE QUALIDADE DE SOFTWARE 2021] Esse projeto foi escolhido por ser um dos mais completos desenvolvidos no grupo em termos de tecnologias e complexidade, possuindo vários subprojetos implementados em tecnologias diferentes. Além disso, atualmente esse é o projeto de software mais ativo desenvolvido no grupo, envolvendo pesquisadores de graduação, mestrado e doutorado.

3. Implementação da Pipeline de CI/CD

Como dito anteriormente, para recolher as informações práticas necessárias para criar um guia para criação de pipelines de CI/CD de qualidade, é necessário implementar, do começo ao fim, um fluxo de integração contínua em um projeto do GQS como exemplo.

O projeto escolhido foi o CodeMaster, por ser um dos projetos mais complexos do laboratório, com maior número de componentes, linguagens diferentes que o compõem e de usuários, conforme já justificado.

O objetivo principal das pipelines de CI/CD do CodeMaster será compilar e executar a aplicação, para que seja validado que o código integrado esteja funcional. Visto isso, é

necessário que a máquina que esteja executando o projeto tenha todos os componentes de software necessários para isto.

Contudo, é inviável que seja configurado todos os requisitos de software do CodeMaster em uma máquina real específica para executar estas pipelines, visto que estes requisitos podem mudar com o tempo, e é possível que seja necessário executar as pipelines em máquinas diferentes, fazendo com que esta opção tenha baixa portabilidade e viabilidade.

Para realizar isto, então, com base no que foi mostrado nos capítulos anteriores, foi criada uma imagem Docker base para o projeto [DOCKER INC. 2022a]. Essa imagem Docker contém todos os requisitos de software necessários para compilar e executar o CodeMaster, e, utilizando a tecnologia Docker, ela se torna completamente portável, e pode ser utilizada em qualquer máquina que tenha o Docker instalado.

Como dito anteriormente, o Docker Hub [DOCKER INC. 2021] é o repositório oficial para imagens Docker. Com ele, é possível guardar imagens Docker para que elas possam ser utilizadas depois em qualquer caso de uso. Por isso, o GQS criou uma conta para o laboratório no Docker Hub para armazenar essas e outras imagens que o laboratório possa ter.

Para criar esta imagem Docker base do CodeMaster, foi criado um arquivo ***Dockerfile*** com os comandos necessários para preparar o ambiente, com base no que foi indicado nas sessões anteriores.

No trecho de código a seguir, é apresentado o ***Dockerfile*** que foi utilizado para a criação da imagem Docker:

```
FROM tomcat:8-jdk8

RUN apt-get update && apt-get install -y \
   maven \
   python3-dev \
   python3-pip \
   nodejs \
   npm \
   git-all \
   ant \
   cmake \
   vim

RUN npm install -g n && n stable

RUN cd ~ && git clone https://github.com/opencv/opencv.git && \
 cd opencv && \
 git checkout 3.4.3 && \
 mkdir build && \
 cd build && \
 export ANT_HOME=/usr/share/ant && \
 cmake .. -DBUILD_SHARED_LIBS=OFF && \
```

```
 make -j8 && \
 mvn install:install-file -Dfile=bin/opencv-343.jar
-DgroupId=org.openpnp -DartifactId=opencv -Dversion=343
-Dpackaging=jar

RUN mkdir -p /srv/codemaster/backend/temp/
RUN mkdir -p /srv/codemaster/backend/projects/
RUN mkdir -p /srv/codemaster/backend/temp/copia_TM
RUN mkdir -p /srv/codemaster/backend/projects/copia_AIA

EXPOSE 8080 8801 4200 3000 8000 9999 3060
```

Este ***Dockerfile*** foi construído e enviado ao Docker Hub, na conta do GQS, e está disponível a acesso por outros pesquisadores no seguinte link: https://hub.docker.com/r/gqsufsc/codemaster.

Tendo a imagem Docker base pronta e acessível através do Docker Hub, bastou implementar a pipeline de CI/CD no projeto CodeMaster. O primeiro passo para implementar a pipeline, foi habilitar a integração e deployment contínuos no projeto. Para isso, foi necessário indicar ao GitLab que os fluxos CI/CD estavam habilitados no projeto, para isso bastou acessar a configuração presente em **Settings > General > Visibility, project features, permissions** da interface do projeto no GitLab, ir até **CI/CD**, e habilitar:

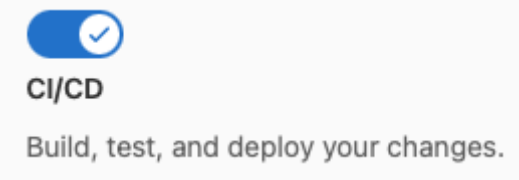


Figura 1. Interface de habilitação do CI/CD

Com apenas esta pequena configuração, o GitLab estava configurado para tentar executar pipelines de CI/CD quando necessário no projeto. Com o CI/CD habilitado no projeto, é necessário configurar um ***Runner*** [GITLAB B.V. 2022b] para executar estas pipelines. Esta configuração é feita no menu **Settings > CI/CD > Runners** da interface do projeto no GitLab.

Olhando esta sessão, é possível ver que o laboratório já possui uma série de **Runners Compartilhados** prontos para serem usados nas pipelines:

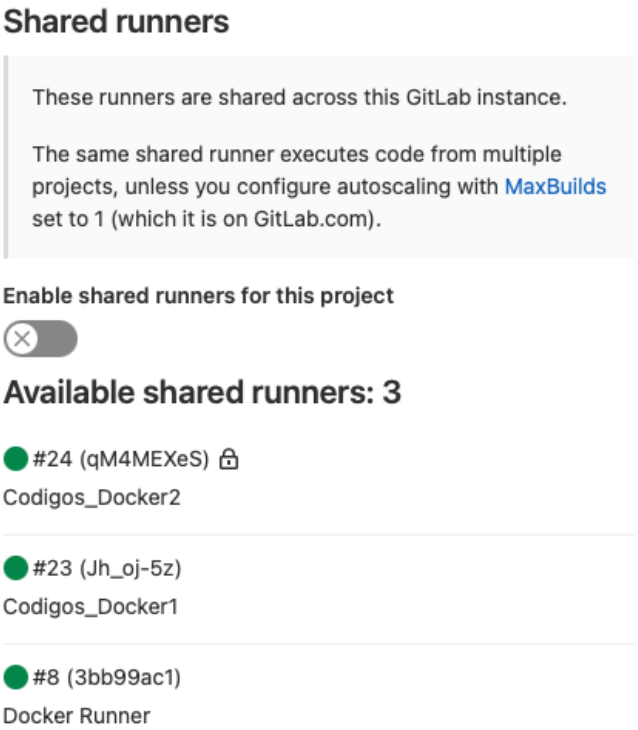


Figura 2. Interface de listagem de runners compartilhados

Estes runners são compartilhados entre os projetos, e estão prontos para serem utilizados e aptos para executar pipelines, porém, eles não foram usados por dois motivos principais:

- O pretexto de criar esta pipeline neste projeto é juntar todas as informações práticas necessárias para a criação do guia de CI/CD do laboratório, por isso, é preciso realizar o processo de configuração completo, incluindo a configuração de um runner do zero.
- Um dos objetivos básicos desta pipeline, seria fazer o *deploy* do projeto para o ambiente de desenvolvimento. Visto isso, é interessante que estas pipelines sejam executadas diretamente pela máquina onde o servidor de desenvolvimento estará rodando, e não por outras máquinas.

Por conta do segundo ponto, então, o runner do projeto foi configurado em uma máquina virtual de desenvolvimento fornecida pela SeTIC, e toda a pipeline do projeto é rodada diretamente na máquina.

Esta máquina de desenvolvimento foi acessada através da máquina pessoal do aluno utilizando ***ssh***. Com isso, foi possível instalar a aplicação gitlab-runner [GITLAB B. V. 2022b] nesta máquina, e utilizar o comando ***gitlab-runner register*** para registrar a máquina como um runner para executar as pipelines do projeto.

Ao rodar este comando, algumas informações foram necessárias para configurar o runner e adicioná-lo ao projeto, sendo elas:

- **URL e token do projeto do GitLab**: Estas são as duas informações que irão relacionar o Runner a um projeto do GitLab.

É possível achar essas informações na página **Settings > CI/CD > Runners**, na aba **Specific runners**:

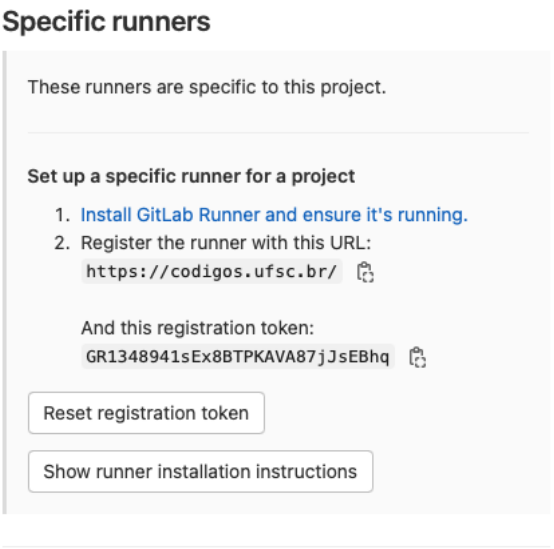


Figura 3. Interface de configuração de runners específicos

- **Executor:** Um ***Runner*** deve possuir um executor [GITLAB B. V. 2022b], que indica qual a maneira que o runner irá executar a pipeline.

O executor **docker** roda as pipelines em uma imagem Docker especificada pelo desenvolvedor, esta imagem Docker gerada é apagada ao fim da execução da pipeline, ou seja, ao fim da execução, tudo o que foi feito é apagado. Já o executor **shell** irá executar a pipeline direto na máquina, qualquer comando que for executado na pipeline, terá efeito diretamente na máquina de testes.

Estes dois executores foram configurados para que possa ser realizada uma experimentação entre diferentes modos de execução de pipelines.Após a configuração ter sido feita, os runners estavam aptos para interpretar e executar as pipelines.

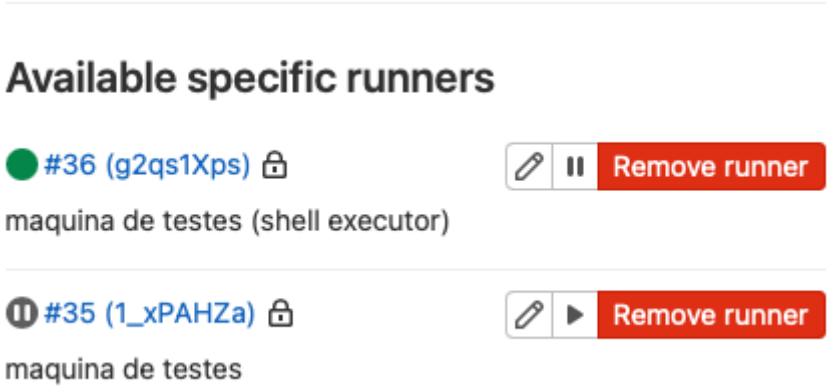


Figura 4. Lista de Runners configurados no CodeMaster

Após um breve estudo e testes preliminares, foi identificado que o executor que seria utilizado da pipeline seria o **shell**, isso por que é necessário que os comandos da pipeline

sejam executados diretamente na máquina de desenvolvimento, e persistidos após o fim do processo, para que o servidor de testes seja levantado para uso dos pesquisadores do grupo GQS.

Contudo, a máquina de desenvolvimento ainda não tem todas as dependências de software necessárias para rodar o CodeMaster, e nem deveria, visto que a imagem Docker base do projeto foi criada para evitar que seja necessário instalar estas dependências de software na máquina de desenvolvimento.

Foi necessário, então, achar uma forma de utilizar a facilidade que a imagem Docker base do CodeMaster proporciona, e ainda rodar a pipeline diretamente na máquina de testes. Para isso, foi criado então alguns ***Dockerfiles*** para o projeto.

Criar ***Dockerfiles*** para o projeto é uma técnica bastante utilizada em fluxos de integração contínua atuais. Ela consiste em criar um ou mais arquivos ***Dockerfile*** na raiz do projeto, que utilizam alguma imagem Docker como base, e possuem toda a lógica para construir e executar o projeto.

Então, quando estes ***Dockerfiles*** são transformados em imagens e executados, eles executam o projeto em questão. Desta maneira, qualquer máquina pode executar o projeto, bastando ter o Docker instalado na máquina em questão, e executar os ***Dockerfiles*** do projeto.

No contexto do CodeMaster, foi necessário criar três dockerfiles diferentes:

- **Dockerfile.backend:** Que constrói e executa todos os componentes de backend do projeto.
- **Dockerfile.frontend:** Que constrói e executa os componentes de frontend do projeto.
  Dockerfile.database: Que inicia um banco de dados SQL de testes para ser utilizado pela aplicação que está executando.

O diagrama a seguir demonstra o desenho final do planejamento do servidor de testes:

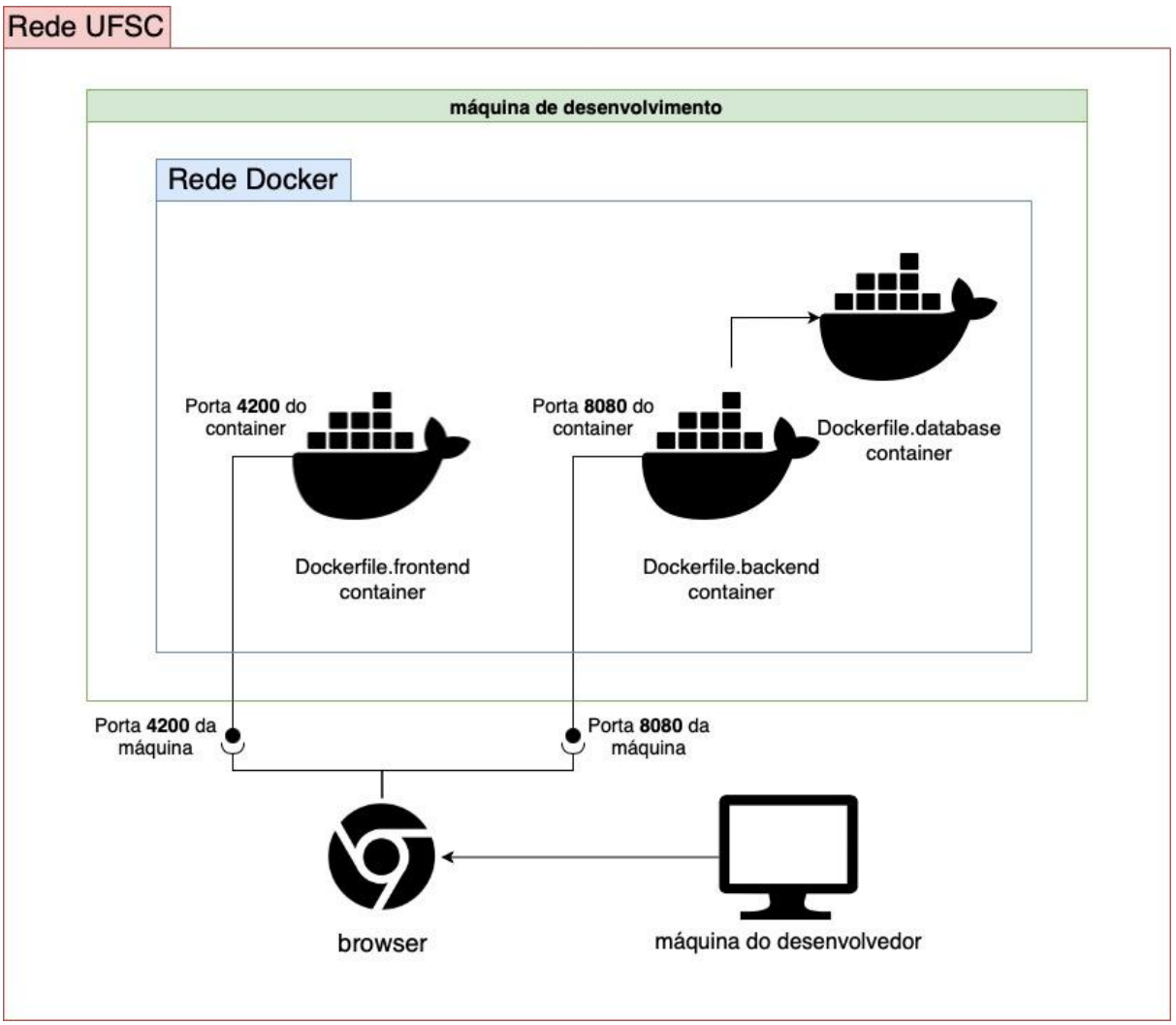


Figura 5. Diagrama de funcionamento do servidor de desenvolvimento

A máquina de desenvolvimento estará rodando os containers de frontend, backend e de banco de dados, que estão inseridos na mesma Rede Docker [DOCKER INC. 2022b] para o CodeMaster. Os containers de backend e frontend estarão expostos para o ambiente externo, e o container de backend irá chamar o container de banco de dados.

A máquina do desenvolvedor, e a máquina de desenvolvimento, deverão estar na mesma rede, que é a rede fornecida pela UFSC. Desta forma, o browser da máquina do desenvolvedor poderá acessar tanto o backend quando o frontend, que estarão rodando em portas específicas da máquina de desenvolvimento.

Com isso, a aplicação será acessível por qualquer máquina presente na rede da UFSC, através dos links **http://<ENDEREÇO IP>:4200** (para o componente web) e **http://<ENDEREÇO IP>:8080** (para o componente de backend), onde **<ENDEREÇO IP>** é o ip da máquina virtual de desenvolvimento.

Com o funcionamento do servidor planejado, a pipeline de CI/CD pode ser criada sem muitos problemas, bastando utilizar a ferramenta Docker para que ela funcione corretamente.

O trecho de código a seguir representa o arquivo ***.gitlab-ci.yml*** que foi criado para implementar as pipelines do projeto CodeMaster:

```
stages:
  - build
  - test
  - stop_test

build database:
  stage: build
  script:
    - sudo docker build -f Dockerfile.database --rm -t codemaster-database .
build frontend:
  stage: build
  script:
    - sudo docker build -f Dockerfile.frontend --rm -t codemaster-frontend .
build backend:
  stage: build
  script:
    - sudo docker build -f Dockerfile.backend --rm -t codemaster-backend .

start test server:
  only:
    - dev
  stage: test
  before_script:
    - sudo docker rm -f codemaster_db || true
    - sudo docker rm -f codemaster_back || true
    - sudo docker rm -f codemaster_front || true
    - sudo docker network rm -f codemaster || true
  script:
    - sudo docker network create codemaster
    - sudo docker run --network codemaster --name codemaster_db -d
      -p 3306:3306 codemaster-database
    - sudo docker run --network codemaster --name codemaster_back -d
      -p 8080:8080 codemaster-backend
    - sudo docker run --network codemaster --name codemaster_front -d
      -p 4200:4200 codemaster-frontend
  environment:
      name: staging
      url: http://150.162.6.91:4200

stop test server:
  stage: stop_test
  script:
```

```
    - sudo docker rm -f codemaster_db || true
    - sudo docker rm -f codemaster_back || true
    - sudo docker rm -f codemaster_front || true
    - sudo docker network rm -f codemaster || true
  when: manual
  environment:
    name: staging
    action: stop
```

Esta pipeline é composta pelos seguintes ***stages*** e ***steps:***

- *Stage* **build**, que é responsável por construir a aplicação, com os steps:
    - **build database:** Constrói o arquivo ***Dockerfile.database*** para gerar a imagem Docker de banco de dados do projeto;
    - **build frontend:** Constrói o arquivo ***Dockerfile.frontend*** para gerar a imagem Docker do componente web do porjeto;
    - **build backend:** Constrói o arquivo ***Dockerfile.backend*** para gerar a imagem Docker dos componentes de back-end do projeto.

- *Stage* **test,** que é responsável por iniciar o servidor de testes, com o step:
    - **start test server:** Inicia a rede Docker e executa os as imagens na rede criada, gerando containers Docker, expondo na máquina de testes as rotas especificadas.
      Este step utiliza o atributo ***before_script*** para definir um script que será gerado antes do step começar, que remove qualquer contêiner que já esteja rodando na máquina.
      Este step também utiliza o atributo ***environment*** para indicar que ele sobe o servidor no ambiente de ***staging***, na url especificada.
      É utilizado também o atributo ***only*** para especificar que este fluxo deverá rodar apenas para a ***branch*** dev do CodeMaster. Esta branch é destinada ao código de desenvolvimento, e é apenas quando for integrado algo nela, que o servidor de desenvolvimento será executado.

- *Stage* **stop_test**, que é responsável por parar o servidor de testes, com o step:
    - **stop test server:** Remove os containers Docker do CodeMaster que estiverem rodando na máquina para encerrar o servidor de testes.
      Este step também utiliza o atributo ***environment***, com a ***action stop***, para indicar que ele encerra o servidor no ambiente de ***staging***.
      Ele também utiliza o atributo ***when*** para indicar que o job é rodado apenas manualmente.

Ao criar o arquivo ***gitlab-ci.yml*** e integrá-lo ao código, a pipeline é executada, e seu resultado pode ser verificado na sessão ***CI/CD > Pipelines*** na página do projeto no GitLab:

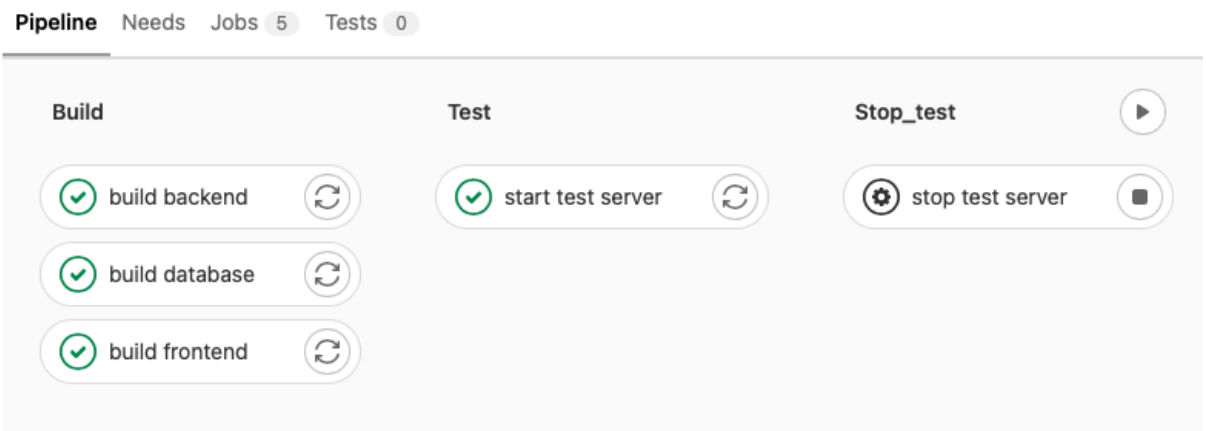
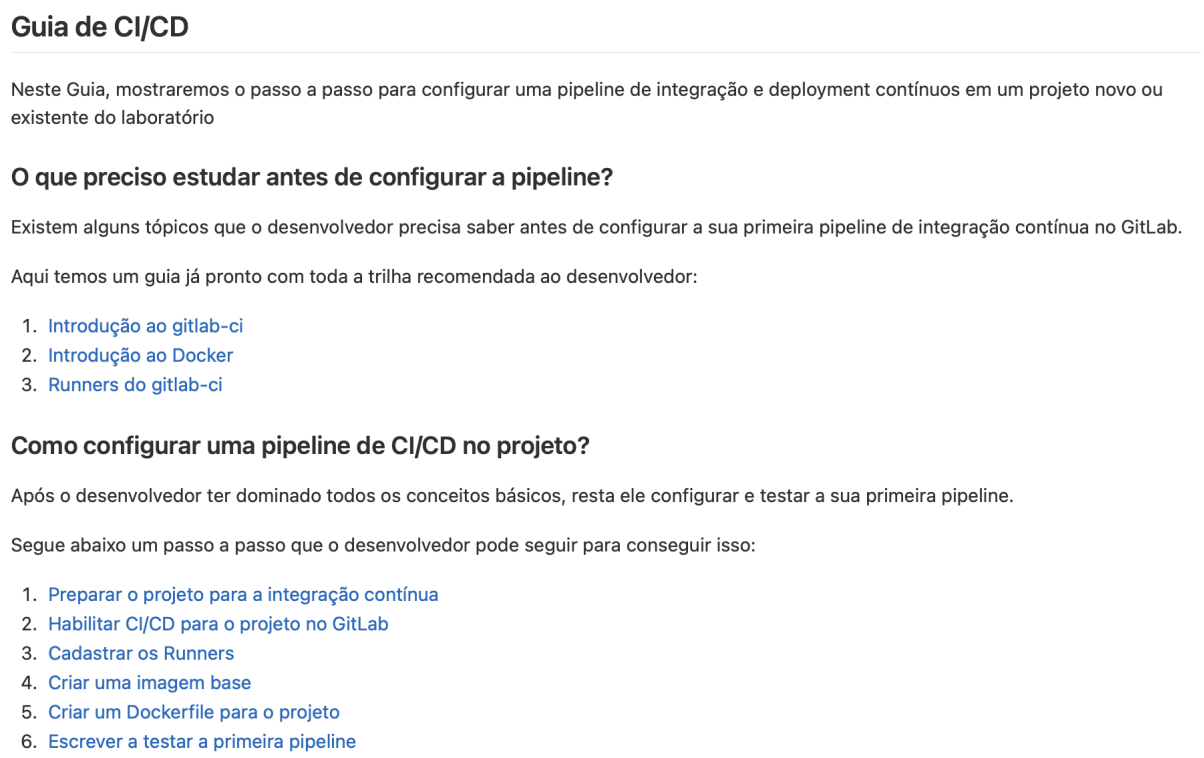


Figura 6. Resultado da pipeline de CI/CD

Como é possível ver na figura 19, os ***stages*** são executados na mesma ordem que foi definida no atributo ***stages*** do arquivo pipeline. Pode-se verificar também que o ***step stop test server*** está com um ícone diferente, que indica que ele deve ser executado manualmente.

4. Criação do Guia de CI/CD

Com o processo exemplo de CI/CD criado no projeto CodeMaster, todas as informações práticas necessárias foram coletadas para a criação de um guia completo de CI/CD para o GQS. Este guia foi criado na ***wiki*** da página inicial do laboratório GQS no GitLab UFSC: https://codigos.ufsc.br/gqs.



Guia de CI/CD

Neste Guia, mostraremos o passo a passo para configurar uma pipeline de integração e deployment contínuos em um projeto novo ou existente do laboratório

O que preciso estudar antes de configurar a pipeline?

Existem alguns tópicos que o desenvolvedor precisa saber antes de configurar a sua primeira pipeline de integração contínua no GitLab.

Aqui temos um guia já pronto com toda a trilha recomendada ao desenvolvedor:

1. Introdução ao gitlab-ci
2. Introdução ao Docker
3. Runners do gitlab-ci

Como configurar uma pipeline de CI/CD no projeto?

Após o desenvolvedor ter dominado todos os conceitos básicos, resta ele configurar e testar a sua primeira pipeline.

Segue abaixo um passo a passo que o desenvolvedor pode seguir para conseguir isso:

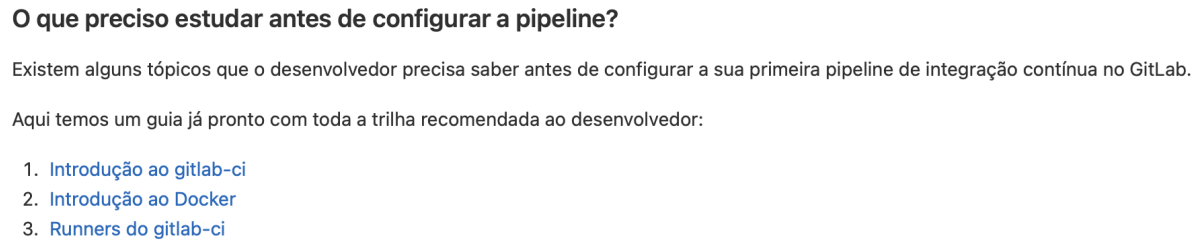
1. Preparar o projeto para a integração contínua
2. Habilitar CI/CD para o projeto no GitLab
3. Cadastrar os Runners
4. Criar uma imagem base
5. Criar um Dockerfile para o projeto
6. Escrever a testar a primeira pipeline

Figura 7. Página inicial do guia de CI/CD do GQS

O objetivo principal deste guia é trazer as informações necessárias ao desenvolvedor de maneira amigável, de forma que qualquer membro do grupo GQS possa entender como as tecnologias utilizadas pelo ***gitlab-ci*** funcionam, e criar suas próprias pipelines de integração contínua usando a ferramenta. Nas seções seguintes, serão apresentados cada um dos tópicos e subtópicos do guia.

4.1 Tópico 1: O Que Estudar

Antes de começar a configurar um guia de CI/CD, existem alguns conceitos básicos que o desenvolvedor precisa dominar ou, pelo menos, entender.


O que preciso estudar antes de configurar a pipeline?

Existem alguns tópicos que o desenvolvedor precisa saber antes de configurar a sua primeira pipeline de integração contínua no GitLab.

Aqui temos um guia já pronto com toda a trilha recomendada ao desenvolvedor:

1. Introdução ao gitlab-ci
2. Introdução ao Docker
3. Runners do gitlab-ci

Figura 8. Tópico 1: o que estudar, do guia de CI/CD do GQS

Este tópico apresenta uma trilha de guias informativos introdutórios que o desenvolvedor deverá seguir a fim de obter as informações teóricas necessárias para configurar a sua primeira pipeline de CI/CD.

Estes conhecimentos requeridos poderiam apenas ser listados, e seria possível referenciar a documentação oficial de cada ferramenta para que os desenvolvedores pudessem pesquisar e ler mais sobre.

Entretanto, estas documentações oficiais, além de geralmente estarem em inglês, possuem muito mais informações do que o necessário para que o desenvolvedor saiba o básico para poder criar sua primeira pipeline.

Visto isso, o objetivo deste tópico é reunir toda e qualquer informação que o desenvolvedor precise saber para que possa criar sua primeira pipeline, juntando essas informações de uma maneira simples e de fácil entendimento.

4.1.1 Gitlab-ci

Este subtópico possui um guia introdutório que explica ao desenvolvedor como funciona a ferramenta gitlab-ci.

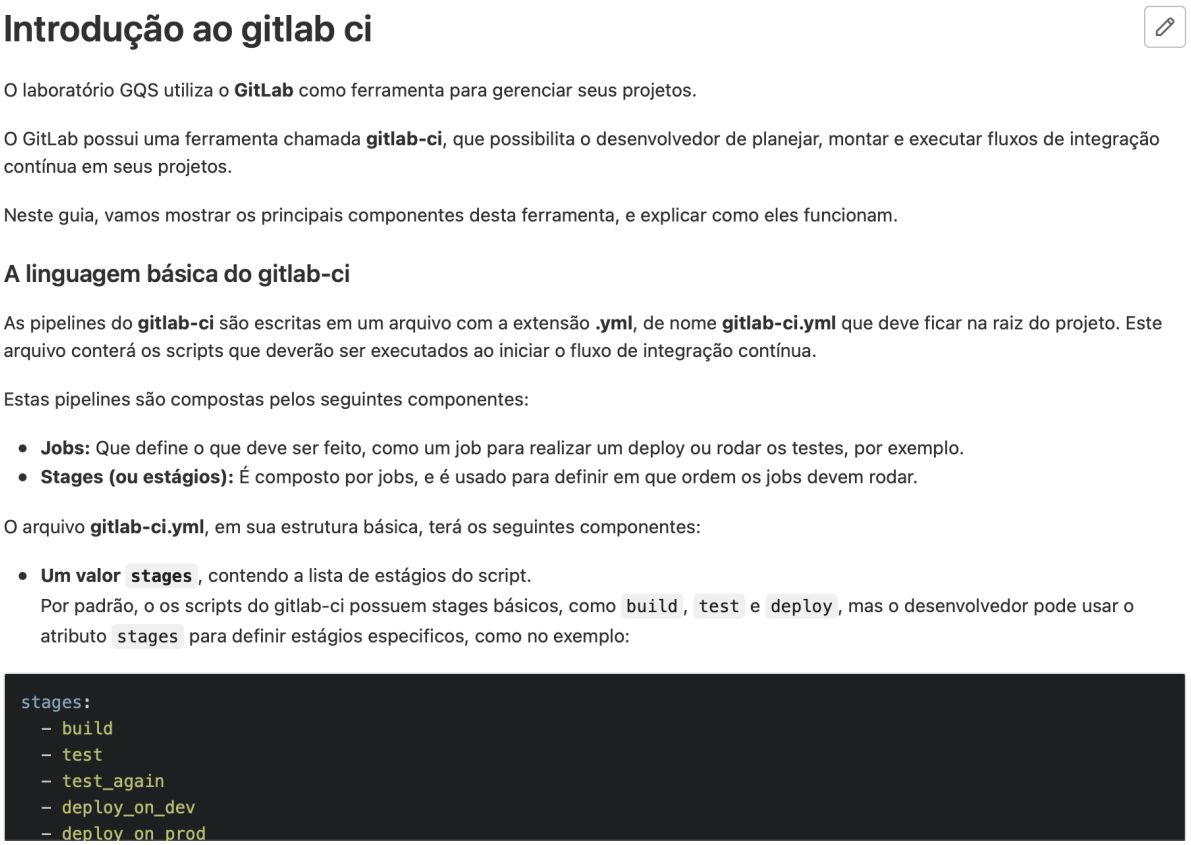


Figura 9. Guia de introdução ao gitlab-ci, página inicial

O gitlab-ci é a ferramenta principal das pipelines de CI/CD do GitLab, sendo ela o motor no qual as pipelines são construídas. Os desenvolvedores precisam ao menos entendê-la para que possam começar a escrever as suas pipelines.

Este tópico agrega todas as informações básicas sobre a ferramenta, se baseando nas informações que foram apresentadas no capítulo 2.6 deste trabalho, como por exemplo: o que são ***jobs*** e ***stages***, como as pipelines devem ser escritas, e quais os principais componentes de uma pipeline de CI/CD.

4.1.2 Docker

Este subtópico possui um guia introdutório que explica ao desenvolvedor como funciona a ferramenta Docker.

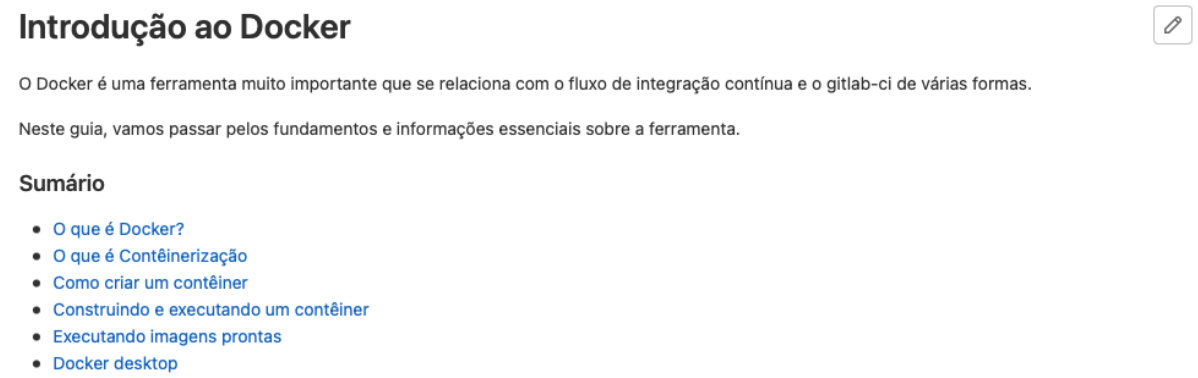


Figura 10. Guia de introdução ao Docker (introdução)

O Docker, como dito anteriormente, é também uma ferramenta muito importante para os fluxos de CI/CD, visto que ela possui uma série de ferramentas que facilitam muito a administração de ambientes de desenvolvimento, e de execução de projetos.

Além disso, a ferramenta Docker se relaciona bastante com as pipelines de CI/CD do gitlab-ci, visto a capacidade das pipelines de utilizar imagens Docker para rodar os comandos das pipelines.

Este tópico visa explicar a fundo o que é o Docker, e introduzir de maneira simples ao desenvolvedor o conceito de conteinerização. É mostrado também como a linguagem Docker funciona e quais seus componentes principais, possuindo uma série de exemplos práticos de como criar Dockerfiles e construir imagens para que o desenvolvedor possa entender, na prática, como esta ferramenta funciona.

4.1.2 Runners

Este subtópico possui um guia introdutório que explica ao desenvolvedor como funcionam os Runners do gitlab-ci.

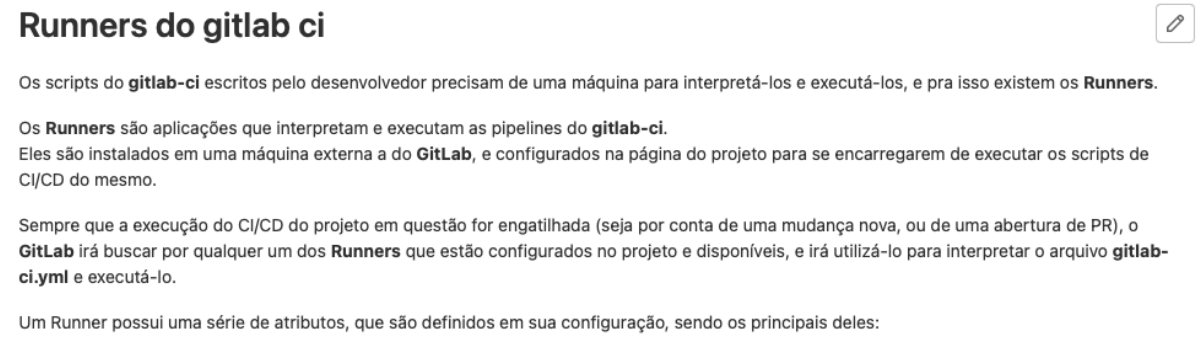


Figura 11. Guia de introdução aos Runners do GitLab, página inicial

Como dito anteriormente, os Runners são uma parte integral e muito importante das pipelines de integração contínua. É preciso entender primeiramente o que são runners e quais suas configurações necessárias antes que o desenvolvedor possa criar a sua pipeline.

Visto isso, esse tópico se baseia no capítulo 2.6.2 deste trabalho para mostrar ao leitor como os runners funcionam, quais são seus atributos, quais são seus executores possíveis e, principalmente, qual o executor preferível no contexto do laboratório GQS.

4.2 Tópico 2: Como Configurar a Pipeline de CI/CD

Com o tópico de introdução concluído, o desenvolvedor deve possuir toda a fundamentação teórica necessária para configurar a pipeline de CI/CD, e, além disso, entender o que está configurando. Este tópico reúne todas as informações apresentadas na fundamentação teórica para indicar um passo a passo que o desenvolvedor pode seguir para ter a sua primeira pipeline de CI/CD rodando no projeto.

4.2.1 Preparar o projeto para a integração contínua

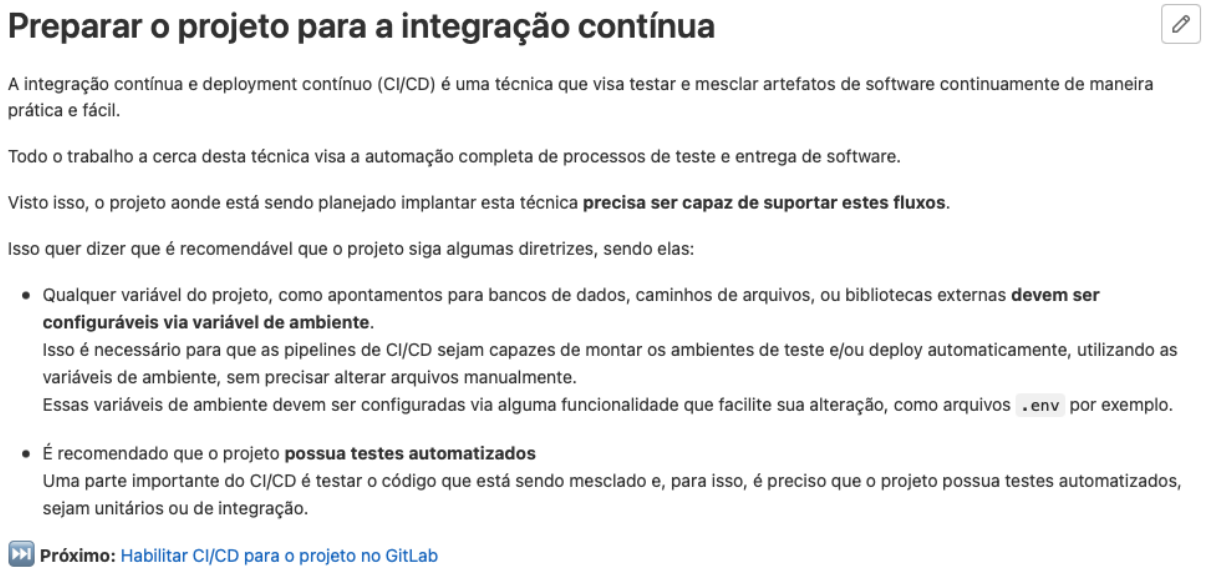


Figura 12. Guia de preparação do projeto para CI/CD

O fluxo de CI/CD, como dito anteriormente, é um fluxo que visa testar e mesclar artefatos de software continuamente e automaticamente, para manter uma qualidade e consistência na entrega de um projeto. Entretanto, para que um projeto seja capaz de implementar uma pipeline de CI/CD ideal e completa, é recomendável que ele possua algumas características chave.

Este subtópico apresenta algumas dessas características, sendo elas:

- **Possuir configuração via variável de ambiente:**
  Isto é recomendável para que a pipeline de CI/CD possa ser executada independente de quaisquer configurações manuais no projeto, como foi feito para o CodeMaster, bastando realizar configurações na própria pipeline para definir apontamentos e outras variáveis.

- **Possuir testes automatizados.**
  Isto é recomendável para que, sempre que a pipeline for executada, o código que for integrado passe por uma bateria de testes que validem regras de negócios e funcionamentos da aplicação.

4.2.2 Habilitar CI/CD no Projeto

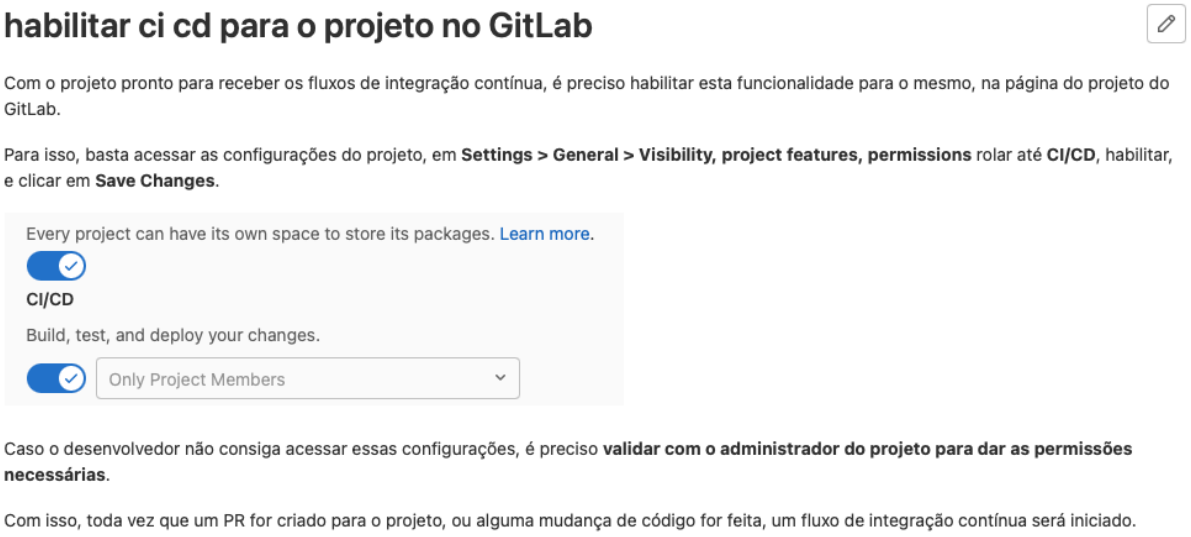


Figura 13. Guia de habilitação de CI/CD no projeto

Para que o fluxo de CI/CD seja habilitado no projeto, é preciso primeiro realizar uma configuração simples na página do projeto no GitLab. Este breve tópico indica qual esta configuração, e onde encontrá-la.

4.2.3 Cadastrar os Runners

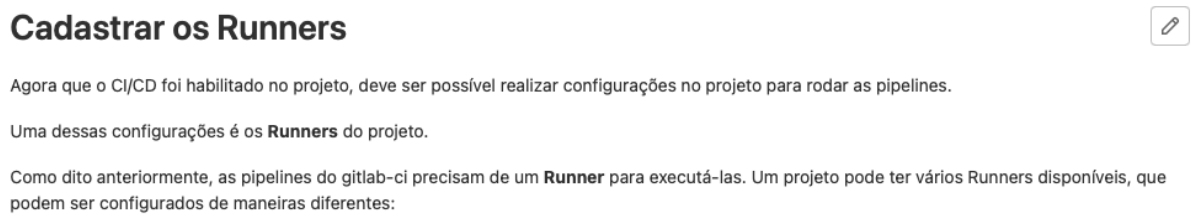


Figura 14. Guia de cadastro de Runners (introdução).

O próximo passo para configurar a pipeline de CI/CD é configurar um Runner para o projeto. Este tópico explica ao desenvolvedor como realizar a configuração deste Runner na máquina de testes do GQS disponibilizada pela SeTIC, se baseando no que foi feito para o CodeMaster no capítulo 4.1.3.2 deste trabalho.Este tópico também apresenta uma maneira alternativa de configurar estes Runners, como é possível ver na figura 15.

Figura 15. Guia de cadastro de Runners (runners do GitLab).

Os Runners providos pelo GitLab são Runners que já estão disponíveis para qualquer projeto do laboratório, que podem ser utilizados pelos desenvolvedores em casos de uso mais simples.

4.2.4 Criar Imagens Base Para o Projeto

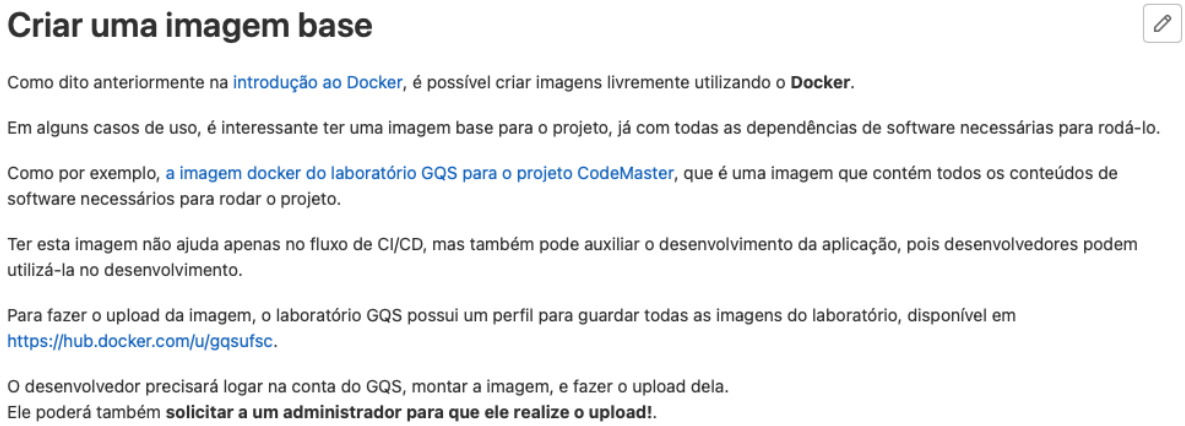


Figura 16. Guia de criação de imagens base (introdução).

Da mesma forma que foi feito para o CodeMaster, é interessante para a maioria dos projetos a criação de uma imagem Docker base que já possua todos os requisitos de software necessários para rodar a pipeline de integração contínua. Este tópico explica ao desenvolvedor não só como criar esta imagem Docker base, mas também como realizar o upload dela para a página do laboratório GQS no Docker.

1. ter o docker instalado na máquina
2. Clonar o projeto https://codigos.ufsc.br/gqs/ci-cd-gqs e adicionar um novo arquivo Dockerfile.
3. Escrever a lógica necessária para preparar a imagem neste Dockerfile, com base no que foi explicado na introdução ao Docker. Para este passo, o desenvolvedor poderá utilizar como base (no comando `FROM`) uma das imagens base do laboratório GQS.
   Pois o laboratório já contém uma série de imagens pré-prontas que pode ser úteis como ponto de partida para criar uma imagem nova.
4. Fazer login no docker, executando o comando `sudo docker login -p {SENHA} -u {USUARIO}`;
   Nota: Conseguir o usuário e senha do docker-hub do GQS com o administrador do laboratório!
5. Construir a imagem docker: `sudo docker build --rm -t gqsufsc/my-image:latest .`;
   Aonde `my-image` é o nome que o desenvolvedor quer dar para a imagem.
6. Fazer push da imagem: `sudo docker push gqsufsc/my-image:latest`.
7. Integrar este novo Dockerfile na branch `main` do projeto https://codigos.ufsc.br/gqs/ci-cd-gqs

Figura 17. Guia de criação de imagens base (passo a passo de criação da imagem Docker).

4.2.5 Criar Um Dockerfile Para o Projeto

Criar um Dockerfile para o projeto

Outra facilidade que o desenvolvedor pode implementar, é escrever um **Dockerfile** para o projeto.

Este **Dockerfile** ficará encarregado de executar a aplicação. Isto facilita muito o fluxo de CI/CD e de desenvolvimento, pois para construir e rodar a aplicação basta utilizar a ferramenta **Docker**.

Para realizar isso, basta o desenvolvedor criar um arquivo `Dockerfile` na raiz do projeto, com base no que foi indicado na Introdução ao Docker. Como imagem base para este Dockerfile (comando `FROM`) ele poderá utilizar uma das imagens bases do GQS ou a imagem base que ele criou no passo anterior!

E então a aplicação pode ser rodada com os comandos:

- `sudo docker build --rm -t my-image .`, para construir a imagem;
- `sudo docker run my-image`, para rodar a imagem.

Algo que poderá ser feito tanto em tempo de desenvolvimento, quanto na pipeline de CI/CD.

Figura 18. Guia de criação de Dockerfiles.

Como também foi feito para o CodeMaster, é interessante para os fluxos de CI/CD que o projeto possua um Dockerfile na sua raiz que possua toda a lógica de execução do projeto, e seja capaz de executá-lo.

Este tópico explica ao desenvolvedor brevemente qual a motivação de realizar esta configuração, indicando como criar um Dockerfile, e como construí-lo em uma imagem Docker, para que seja executada em um container.

4.2.5 Escrever e testar a primeira pipeline

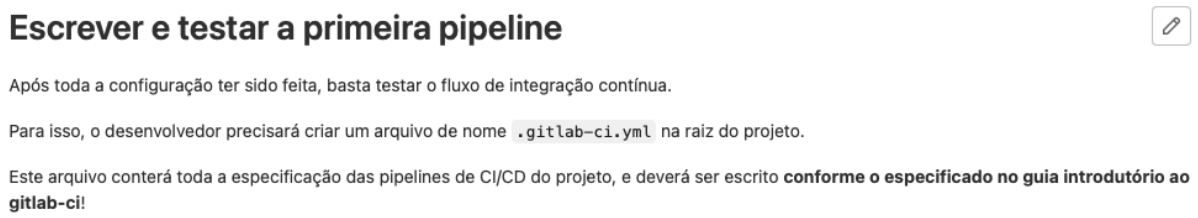


Figura 19. Guia de criação da primeira pipeline de CI/CD.

Por fim, tendo toda a configuração pronta, este tópico indica ao desenvolvedor como criar e testar a sua primeira pipeline simples de CI/CD. Este tópico apresenta também uma pipeline de "modelo" funcional, para que os desenvolvedores possam ter um ponto de partida e possam testar as configurações feitas, sendo ela:

```
image: gqsufsc/codemaster:latest

stages:
  - build
  - test
  - deploy

build application:
  stage: build
  script:
    - echo "BUILD APPLICATION!"

test application:
  stage: test
  script:
    - echo "TEST APPLICATION!"

deploy application:
  stage: deploy
  script:
    - echo "DEPLOY APPLICATION!"
```

Esta pipeline possui três estágios básicos que executam comandos simples. Os desenvolvedores podem utilizá-la para validar se as configurações foram feitas corretamente.

4.3 Tópicos Avançados

Por fim, como um pequeno extra, é apresentado também ao final do guia alguns tópicos extras avançados no contexto das tecnologias e ferramentas que foram apresentadas.

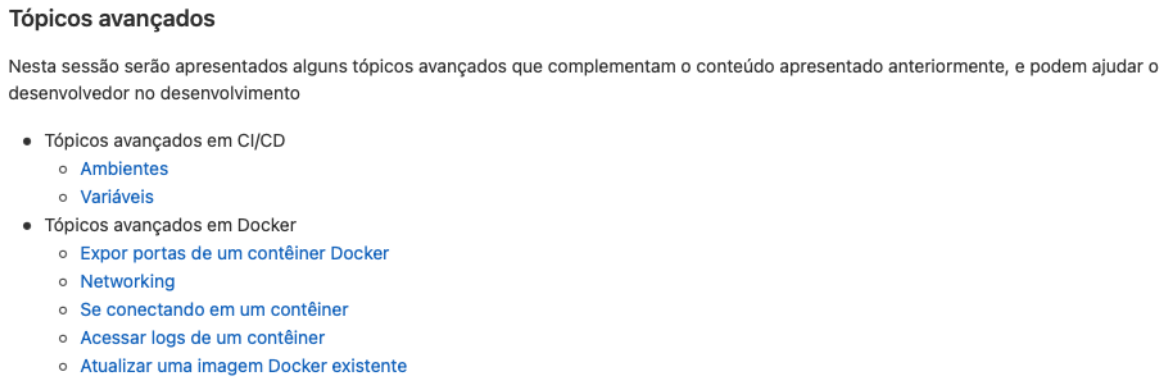


Figura 20. Tópicos avançados do guia de CI/CD do GQS.

Estes tópicos foram criados a fim de ajudar os desenvolvedores em casos de uso mais complexos das ferramentas ***gitlab-ci*** e ***Docker***, reunindo todas as dificuldades que o autor deste trabalho passou ao criar a pipeline de CI/CD do CodeMaster, a fim de evitar que novos desenvolvedores passem pelos mesmos empecilhos.

5. Conclusão

Como é possível observar nos resultados deste projeto, implantar o conceito de integração e deployment contínuos em um time de software pode trazer benefícios além dos que foram previamente identificados na fundamentação teórica deste trabalho.

Claramente, as pipelines de CI/CD ajudam muito no desenvolvimento de um produto de software ao automatizar todo o processo de teste e entrega do mesmo. Processo este que pode vir a ser difícil de fazer, à medida que o time, e o produto de software, crescem.

Porém, pode-se identificar outros benefícios. Ao implantar o conceito de CI/CD, o time responsável pelo produto de software acaba automaticamente adotando boas práticas que facilitam, ou até possibilitam, a implantação das pipelines de integração contínua. Como, por exemplo, o uso do Docker, uso de variáveis de ambiente e configurações remotas, a adoção de testes automatizados, entre outros.

Todas estas boas práticas que estão dentro do conceito de CI/CD melhoram significativamente a qualidade de um produto de software, bem como a facilidade a qual ele é configurado, desenvolvido, e distribuído.

Isso pôde ser identificado na implantação do fluxo de CI/CD no projeto CodeMaster, visto que era um projeto complexo de difícil manutenção, onde desenvolvedores gastavam uma grande quantidade de tempo para configurá-lo antes de conseguir trabalhar no mesmo.

Agora, ao implantar a integração contínua, o projeto possui uma configuração muito mais fácil e amigável, utilizando os mesmos Dockerfiles que a pipeline de CI/CD utiliza.

Com isso, pode ser identificada a importância deste projeto. A UFSC e o GQS possuem projetos de grande importância no âmbito social e, manter a qualidade da entrega destes projetos é algo de extrema importância.

Ao implantar o conceito de integração e *deployment* contínuos no GQS, espera-se que seja também implantada uma nova maneira de planejar, desenvolver, e entregar software no laboratório, com melhores práticas de desenvolvimento, maior qualidade nas entregas e uma maior facilidade no processo de entrega destas aplicações.

5. Referências

DOCKER INC. **Docker Hub,** 2021. Hub de imagens do docker.
Disponível em: <https://hub.docker.com>. Acesso em 29 de junho de 2022.

DOCKER INC. **Docker Hub,** 2022a. Imagem base do projeto CodeMaster.
Disponível em: <https://hub.docker.com/r/gqsufsc/codemaster>.
Acesso em 07 de novembro de 2022.

DOCKER INC. **Networking Overview,** 2022b. Documentação oficial das redes Docker.
Disponível em: <https://docs.docker.com/network/>.
Acesso em 08 de novembro de 2022.

GÓIS, F. et. al. Desenvolvimento de Software Utilizando Integração Contínua.
ERCEMAPI 2007, Ceará, p. 62-80, jul. 2007.
Disponível em: <https://www.researchgate.net/profile/Pedro-Muniz-Farias/publication/353273690_ERCEMAPI_2007_versao_final_completa_com_isbn/links/60f0890c16f9f3130087493b/ERCEMAPI-2007-versao-final-completa-com-isbn.pdf#page=62>.

GITLAB B. V. **GitLab**, 2022a. Página inicial.
Disponível em <https://about.gitlab.com>.
Acesso em 28 de junho de 2022.

GITLAB B. V. **Install GitLab Runner**, 2022b.
Documentação oficial do GitLab referente a instalação de runners.
Disponível em: <https://docs.gitlab.com/ee/ci/pipelines/index.html>.
Acesso em 28 de junho de 2022.

GRUPO DE QUALIDADE DE SOFTWARE. **GQS - Software Quality Group**, 2022.
Página inicial. Disponível em: <http://www.gqs.ufsc.br>.
Acesso em 28 de junho de 2022.

GRUPO DE QUALIDADE DE SOFTWARE. **Codemaster,** 2021.
Página inicial do projeto CodeMaster.
Disponível em: <http://apps.computacaonaescola.ufsc.br/codemaster/>.
Acesso em 29 de junho de 2022.

IEEE STANDARDS ASSOCIATION. **IEE 2675-2021:** ***IEEE Standard for DevOps: Building Reliable and Secure Systems Including Application Build, Package, and Deployment***. Fev. 2021.
Disponível em: <https://standards.ieee.org/ieee/2675/6830/>.

PINTO, A; TERRA, R. **Processo de Conformidade Arquitetural em Integração Contínua.** 2015. Disponível em: <http://professores.dcc.ufla.br/~terra/publications_files/students/2015_ufla_pinto.pdf>.

TERRA, R; TÚLIO, M. **Definição de Padrões Arquiteturais e seu Impacto em Atividades de Manutenção de Software.** 2010. Disponível em: <https://www.researchgate.net/publication/262563350_Definicao_de_Padroes_Arquiteturais_e_seu_Impacto_em_Atividades_de_Manutencao_de_Software>.