

UNIVERSIDADE FEDERAL DE SANTA CATARINA

**Implantação de Integração e Deployment Contínuos no sistema de TCCs do
Departamento de Informática e Estatística**

Pedro Schlickmann Mendes

Florianópolis - SC

2024/2

UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
CURSO DE SISTEMAS DE INFORMAÇÃO

**Implantação de Integração e Deployment Contínuos no sistema de TCCs do
Departamento de Informática e Estatística**

Pedro Schlickmann Mendes

Trabalho de conclusão de curso apresentado
como parte dos requisitos para obtenção do grau de
Bacharel em Sistemas de Informação

Florianópolis - SC

2024/2

Pedro Schlickmann Mendes

**Implantação de Integração e Deployment Contínuos no sistema de TCCs do
Departamento de Informática e Estatística**

Trabalho de conclusão de curso apresentado como parte dos requisitos para obtenção do
grau de Bacharel em Sistemas de Informação

Florianópolis, _____ de _____ de 2024.

Prof. Jean Carlo Rossa Hauck
Orientador
Universidade Federal de Santa Catarina

Prof. Renato Cislighi
Membro da Banca Examinadora
Universidade Federal de Santa Catarina

Prof. Frank Augusto Siqueira
Membro da Banca Examinadora
Universidade Federal de Santa Catarina

RESUMO

O Departamento de Informática e Estatística da Universidade Federal de Santa Catarina utiliza um sistema de software para gerenciar os Trabalhos de Conclusão de Curso (TCCs) e suas disciplinas. A partir da observação das dificuldades na manutenção e evolução do sistema, especialmente pela ausência de práticas de automação no processo de desenvolvimento, identificou-se a necessidade de implementar um processo de Integração e Deployment Contínuos (CI/CD). O objetivo deste trabalho é modernizar o Sistema de TCCs por meio da adoção de práticas de DevOps, incluindo a containerização da aplicação, a configuração de pipelines automatizados no GitLab CI/CD e a criação de uma documentação técnica abrangente. Com base em uma análise de requisitos e adaptação de guias existentes, foi implementado um processo que simplifica o desenvolvimento, reduz erros manuais e aumenta a eficiência do *deploy*. A avaliação realizada indica que as mudanças trouxeram melhorias significativas para a confiabilidade e escalabilidade do sistema.

Palavras-chave: CI/CD. Containerização. *Pipelines*. *Docker*. Manutenção

ABSTRACT

The Department of Informatics and Statistics at the Federal University of Santa Catarina manages final graduation projects (TCCs) and their associated courses through a software system. Observing challenges in the system's maintenance and evolution, particularly due to the absence of automated development practices, the need for implementing a Continuous Integration and Deployment (CI/CD) process was identified. This work aims to modernize the TCC System by adopting DevOps practices, including containerizing the application, configuring automated pipelines in GitLab CI/CD, and developing comprehensive technical documentation. Based on a requirements analysis and adaptation of existing guides, a process was implemented to simplify development, reduce manual errors, and improve deployment efficiency. Evaluations indicate that the changes have significantly enhanced the system's reliability and scalability.

Keywords: CI/CD, Containerization, Pipelines. Docker. Maintenance

LISTA DE ABREVIATURA E SIGLAS

INE - Departamento de Informática e Estatística

UFSC - Universidade Federal de Santa Catarina

TCC - Trabalho de Conclusão de Curso

CI - *Continuous Integration* (Entrega Contínua)

CD - *Continuous Deployment* (Implantação Contínua)

API - Interface de Programação de Aplicativos

REST - *Representational State Transfer*

LXC - *Linux Containers*

CLI - *Command Line Interface*

DNS - *Domain Name System*

NFS - *Network File System*

LISTA DE FIGURAS

Figura 1: Sistema de TCCs do INE	16
Figura 2: Níveis de acesso. (Botelho e Ugioni, 2015)	17
Figura 3: Casos de uso Aluno (Botelho e Ugioni, 2015)	18
Figura 4: Casos de uso Avaliador (Onghero, 2018)	19
Figura 5: Arquitetura de desenvolvimento do modelo de recomendação (Leal, 2022).	20
Figura 6: Comparação das camadas envolvidas ao rodar uma aplicação numa máquina virtual ou em um contêiner (Vaucher, 2015)	21
Figura 7: Arquitetura do Docker (Docker Inc, 2023)	23
Figura 8: Deployment Contínuo (CD)	25
(Fonte: https://developerexperience.io/articles/continuous-delivery)	25
Figura 9: Lista de pipelines executados	26
Figura 10: Etapas de um pipeline	27
Figura 11: Funcionalidades do GitLab	28
Figura 12: Quatro passos da configuração do ambiente de desenvolvimento (Onghero, 2018)	30
Figura 13: Problemas nas dependências do projeto.	31
Figura 14: Configuração de execução utilizada no ambiente de desenvolvimento (Onghero 2018)	32
Figura 15: Deployment Diagram (elaborado pelo Autor)	35
Figura 16: Dockerfile inicial	38
Figura 17: docker-compose.yml inicial	38
Figura 18: Ajustando o host do banco de dados	39
Figura 19: Utilizando o Tomcat	40
Figura 20: Otimizando a construção da imagem	40
Figura 21: Package & Registries habilitado	42
Figura 22: Arquivo .gitlab-ci.yml	43
Figura 23: docker-compose.yml de homologação	44
Figura 24: CI/CD funcionando	45
Figura 25: Mapeamento da proxy reversa	47
Figura 26: Wiki do projeto	48
Figura 27: README do projeto	49
Figura 28: Resultados da pergunta 1	52
Figura 29: Resultados da pergunta 2	53

SUMÁRIO

1	Introdução	10
1.1	Objetivos	12
1.1.1	Objetivos específicos	12
1.2	Método de pesquisa	13
2	Fundamentação Teórica	15
2.1	O Sistema de Gerenciamento de TCCs do INE	15
2.1.1	Papel aluno	17
2.1.2	Papel avaliador	18
2.1.3	Módulo de recomendação de temas de TCC	19
2.2	Containerização	20
2.2.1	Docker	22
2.2.1.1	Servicos e volumes	24
2.3	Integração e Deployment contínuos	24
2.3.1	Pipeline de CI/CD	26
2.3.2	Gitlab	27
3	Proposta de Solução	29
3.1	Análise da situação atual	29
3.1.1	Estrutura de arquivos	29
3.1.2	Ambiente de desenvolvimento	30
3.1.3	Ambiente de produção	32
3.1.4	Levantamento das necessidades de CI/CD	33
3.2	Proposta de CI/CD para o sistema de TCCs	34
3.2.1	Containerização	34
3.2.2	Automação de CI/CD no Gitlab	36
3.2.3	Documentação de DevOps	36
3.2.4	Cobertura da Proposta de Solução	36
3.3	Implementação inicial da containerização	37
3.3.1	Problemas de rede	39
3.3.2	Adaptação do projeto para uso do Tomcat	39
3.3.3	Otimização do ciclo de vida do Maven	40
4	Implementação	41
4.1	Configuração das <i>Pipelines</i> de CI/CD	41
4.1.1	Estruturação das <i>Pipelines</i>	41
4.1.2	Integração com o GitLab CI/CD	42
4.2	Implantação do CI/CD	43
4.2.1	Servidor de homologação	44
4.2.2	Servidor de produção	45
4.2.3	Arquivos estáticos	46
4.2.4	Banco de dados	46
4.2.5	HTTPS e Proxy Reverso	47
4.3	Documentação de DevOps	47
4.3.1	Atualização da Wiki	48
4.3.2	Atualização do README.md	49

4.4	Avaliação	49
4.4.1	Planejamento e Coleta de Dados	50
4.4.2	Resultados	51
4.4.2.1	Como você avaliaria a usabilidade do Guia de DevOps?	52
4.4.2.2	A ordem em que os passos do Guia foram colocados é adequada?	52
4.4.2.3	Principais pontos positivos do pipeline CI/CD e da containerização?	53
4.4.2.4	Principais pontos negativos do pipeline CI/CD e da containerização?	54
4.4.2.5	Sugestões para melhorar o pipeline ou a containerização?	54
4.4.3	Considerações finais dos resultados	55
5	Conclusão	56
5.1	Trabalhos futuros	57
	Referências	58

1 Introdução

Os sistemas de *software* que geram valor para os seus usuários tendem a sofrer mudanças ao longo de sua vida útil (Pressman, 2015). À medida que os sistemas de *software* evoluem e crescem em complexidade, torna-se fundamental manter rigorosamente boas práticas de desenvolvimento de *software*. Estas práticas, quando seguidas, proporcionam uma estrutura sólida para que novas funcionalidades sejam implementadas no sistema com mais facilidade. Justamente nesse contexto de evolução e adaptação contínua do *software* surge a necessidade de abordagens como o DevOps.

DevOps, uma contração das palavras "Desenvolvimento" e "Operações", é um conjunto de princípios e práticas que permitem melhor comunicação e colaboração entre as partes interessadas relevantes para fins de especificação, desenvolvimento e operação de produtos e serviços de *software* e sistemas, além de melhorias contínuas em todos os aspectos do ciclo de vida (IEEE 2675). Integração Contínua (do inglês *Continuous Integration* - CI) e *Deployment* Contínuo (do inglês *Continuous Deployment* - CD) são práticas contempladas pelo DevOps. CI pode ser definida como uma técnica que mescla continuamente artefatos, incluindo atualizações de código-fonte de todos os desenvolvedores de uma equipe, em uma linha principal compartilhada para criar e testar o sistema desenvolvido (IEEE 2675). Já o CD é um processo automatizado de implementação de alterações na produção, verificando os recursos pretendidos e as validações para reduzir o risco (IEEE 2675). Esses conceitos, quando aplicados, são geralmente gerenciados por meio de *pipelines*.

Pipelines de CI/CD são processos automatizados que guiam o código através das etapas de integração (CI) e implantação (CD) (IEEE 2675). Na CI, o *pipeline* verifica o código mais recente com testes automatizados, garantindo sua integridade. Já no processo de CD, o *pipeline* promove a entrega deste código nos ambientes configurados, garantindo uma implantação rápida e livre de erros humanos.

A importância do CD torna-se aparente quando são considerados os riscos associados ao erro humano durante a implantação de um sistema para utilização. Um exemplo é o Knight Capital Group, que perdeu US\$440 milhões em menos de uma hora em 2012 devido a uma falha na implantação: um desenvolvedor não se lembrou de atualizar o código em um dos oito servidores da empresa (Dolfing, 2019). Este erro causou caos nas operações financeiras e levou a perdas catastróficas.

O Departamento de Informática e Estatística (INE) da Universidade Federal de Santa Catarina (UFSC) foi criado em 1970, e produz conhecimento nas áreas de informática e estatística, por meio de dois cursos de graduação (Ciências da Computação e Sistemas de Informação) e de dois programas de pós-graduação (INE, 2023). Nos cursos de Ciências da Computação e de Sistemas de Informação, os alunos realizam Trabalhos de Conclusão de Curso (TCC) ao final do curso.

O Regimento Interno Para Elaboração de Trabalhos de Conclusão de Curso (TCC) (Colegiado de Sistemas de Informação, 2011), define que o TCC é obrigatório para a integralização da grade curricular do curso e é desenvolvido ao longo de uma sequência de três disciplinas, sendo a primeira pré-requisito da segunda e a segunda pré-requisito da terceira. As três disciplinas são: Introdução ao Projeto; Projeto I e Projeto II.

Nesse contexto, foi desenvolvido um Sistema de Gerenciamento de TCCs (<https://tcc.inf.ufsc.br/>) no departamento INE para simplificar diversos processos para o aluno e o orientador do TCC. A submissão da proposta formal, necessária para completar a primeira disciplina de TCC é realizada através do sistema, assim como as submissões para Projetos I e II. Outras funcionalidades como formação da banca de avaliação do trabalho e o convite para orientação também são realizadas no Sistema de TCCs.

O Sistema de TCCs foi desenvolvido como um Trabalho de Conclusão de Curso em 2015, pelos alunos Felipe Gonçalves Botelho e Pedro Henrique Rocha Ugioni, orientados pelos professores Renato Cislighi e Antônio Carlos Mariani (Botelho e Ugioni, 2015). Desde então, o sistema foi evoluindo por meio de diversos trabalhos

(Gonçalves, 2016; Onghero, 2018; Baptista, 2019). No entanto, o desenvolvimento do Sistema de TCCs tem sido realizado sem suporte a práticas de CI/CD, o que dificulta a sua manutenção e evolução.

No contexto da UFSC, os sistemas desenvolvidos adotam o GitLab (GitLab Inc, 2023) como ferramenta padrão para o gerenciamento de versões, garantindo rastreabilidade, colaboração e integridade do código-fonte. Uma das vantagens do GitLab é seu suporte nativo a CI/CD, o que agiliza a configuração de pipelines (GitLab CI/CD, 2023). Outra vantagem de todos os sistemas desenvolvidos na UFSC utilizarem a mesma plataforma é que configurações de pipelines de um sistema podem ser reutilizadas como base em outros sistemas, mesmo que eles sejam totalmente distintos.

Pensando nisso, um trabalho anterior (Andrade, 2022) desenvolveu um guia de CI/CD para que desenvolvedores consigam implementar os fluxos de integração contínua de forma facilitada em projetos já existentes, ou em projetos futuros. Assim, este trabalho procura implantar um processo de CI/CD no Sistema de TCCs da UFSC aplicando e tomando como base o guia de CI/CD desenvolvido.

1.1 Objetivos

O objetivo principal deste trabalho consiste na implantação de um processo de CI/CD para o Sistema de TCCs do INE. O guia de CI/CD existente, desenvolvido em trabalho anterior (Andrade, 2022), será adaptado às características do Sistema de TCCs do INE e incorporado à documentação técnica do sistema, de forma que possa ser utilizado por futuros desenvolvedores do sistema.

1.1.1 Objetivos específicos

- OE1 - Analisar a literatura em relação à implementação de CI/CD
- OE2 - Adaptar o guia de CI/CD para o Sistema de TCCs
- OE3 - Implementar o processo de CI/CD para o sistema de TCCs

- OE4 - Implantar e avaliar o processo implementado

1.2 Método de pesquisa

Nesta seção é delineada a abordagem metodológica que guiará a realização deste Trabalho de Conclusão de Curso. A pesquisa será de natureza aplicada e descritiva. Inicialmente utilizando as técnicas e estratégias estabelecidas por Andrade (2022) no guia de CI/CD previamente elaborado e, posteriormente, criando guias de DevOps para o sistema de TCCs.

Etapa 1: Fundamentação teórica

A1.1 - Estudar a literatura

A1.2 - Escrever e revisar conceitos

Consiste na construção da base teórica, envolvendo o estudo aprofundado da literatura, a redação e a revisão crítica dos conceitos-chave de CI/CD e DevOps.

Etapa 2: Análise do contexto

A2.1 - Analisar as características da aplicação

A2.2 - Analisar as características da infraestrutura

Visa analisar as características específicas da aplicação do TCCs e da infraestrutura tecnológica atual, para identificar as necessidades e requisitos do sistema de *pipelines* de CI/CD.

Etapa 3: Desenvolvimento do Devops

A3.1 - Dockerizar banco de dados

A3.2 - Dockerizar aplicação

A3.3 - Desenvolver *scripts* de automação

Nesta etapa ocorre a execução prática do projeto, que inclui a containerização do banco de dados da aplicação e o desenvolvimento de *scripts* de automação para agilizar e otimizar o processo de implantação.

Etapa 4: Documentação de *DevOps*

A4.1 - Elaborar guias de *DevOps*

Envolve a criação de guias e manuais que documentem o processo de implantação e operacionalização do sistema de CI/CD, garantindo que o conhecimento gerado seja acessível e reutilizável.

Etapa 5: Implantação

A5.1 Implantar em servidor de teste

A5.2 Implantar em servidor de produção

É a fase final onde o sistema é primeiramente implantado em um servidor de teste para avaliação e, posteriormente, em um servidor de produção.

Etapa 6: Avaliação

A6.1 Reunir painel de especialistas

Consiste em reunir 5 especialistas na área de DevOps e CI/CD para avaliar possíveis melhorias na implementação.

2 Fundamentação Teórica

Este capítulo estabelece os fundamentos teóricos indispensáveis à compreensão da proposta deste trabalho, que visa aprimorar o sistema de gestão de Trabalhos de Conclusão de Curso (TCCs) do Departamento de Informática e Estatística da Universidade Federal de Santa Catarina (INE/UFSC) por meio da implementação de práticas de Integração e *Deployment* Contínuos (CI/CD). São discutidos os conceitos fundamentais relacionados às tecnologias de containerização, versionamento de código (GitLab) e *Pipelines* de CI/CD.

2.1 O Sistema de Gerenciamento de TCCs do INE

O Sistema de Gerenciamento de TCCs do Departamento de Informática e Estatística (INE) da Universidade Federal de Santa Catarina (UFSC) representa um marco no gerenciamento de projetos de TCCs dos cursos e foi desenvolvido a partir de um projeto de TCC (Botelho e Ugioni, 2015).

O sistema usa a linguagem Java¹, com o *framework* Spring² para suporte à aplicação e JSF³ para desenvolver a interface do usuário. Para organizar e acessar os dados, ele se apoia no Hibernate⁴, que facilita o mapeamento entre os objetos Java e o banco de dados, enquanto o MySQL⁵ é utilizado como banco de dados relacional.

¹ https://www.java.com/pt_BR

² <https://spring.io>

³ <https://javaee.github.io/javaxserverfaces-spec>

⁴ <http://hibernate.org>

⁵ <https://www.mysql.com>

UNIVERSIDADE FEDERAL DE SANTA CATARINA | Sistema de Gestão de TCCs INE | Pedro Schlickmann Mendes (Sair)

INICIO

- Página Inicial

MEU TCC

- Meu TCC
- Documentos
- Equipe
- Avaliações
- Horários para Apresentação

INFORMAÇÕES

- Lista de TCCs
- Links Interessantes
- Arquivos Úteis
- Dúvidas Frequentes
- Professores
- Calendário
- Apresentações Definidas

Meu TCC

Você pode inserir ou modificar as informações do projeto através dos campos abaixo. Todo projeto precisa ter um professor responsável. Quando você define as informações do projeto pela primeira vez, deve convidar um professor para ser responsável. É fundamental que você já tenha conversado com ele antes de usar este sistema. Isso vai garantir um funcionamento mais rápido do processo de definição do projeto. A definição do seu projeto só será considerada válida quando o professor responsável confirmar sua participação através do sistema.

Alterar

Título:
Implementação de Integração e Deployment Contínuos no sistema de TCCs do Departamento de Informática e Estatística.

Descrição:
Desenvolvimento, configuração e implantação de Integração e Deployment Contínuos no sistema de TCCs do Departamento de Informática e Estatística.

Responsável:
Jean Carlo Rossa Hauck

Informações

TCC | Parceiro | Responsável | **Orientação** | Coordenação | Banca

Nome	Email	Status
 Jean Carlo Rossa Hauck	jean.hauck@ufsc.br	Confirmado
Nenhum membro convite (pendente)		

https://tcc.inf.ufsc.br/projeto/projetoAdicionar.xhtml#_dt153j:_dt154j:_dt228

Figura 1: Sistema de TCCs do INE

O sistema (Figura 1) está acessível no endereço eletrônico <https://tcc.inf.ufsc.br>. O visitante é recebido pela página de boas-vindas do portal. Para explorar as funcionalidades oferecidas é necessário realizar o login utilizando o Sistema de Autenticação Centralizada da UFSC. Os usuários que interagem com o sistema são categorizados em dois grupos principais, com privilégios distintos: alunos e avaliadores. Sendo que podem ser adicionados a esse último as permissões de administrador do curso, administrador geral e possível responsável (Botelho e Ugioni, 2015), conforme mostra a Figura 2:

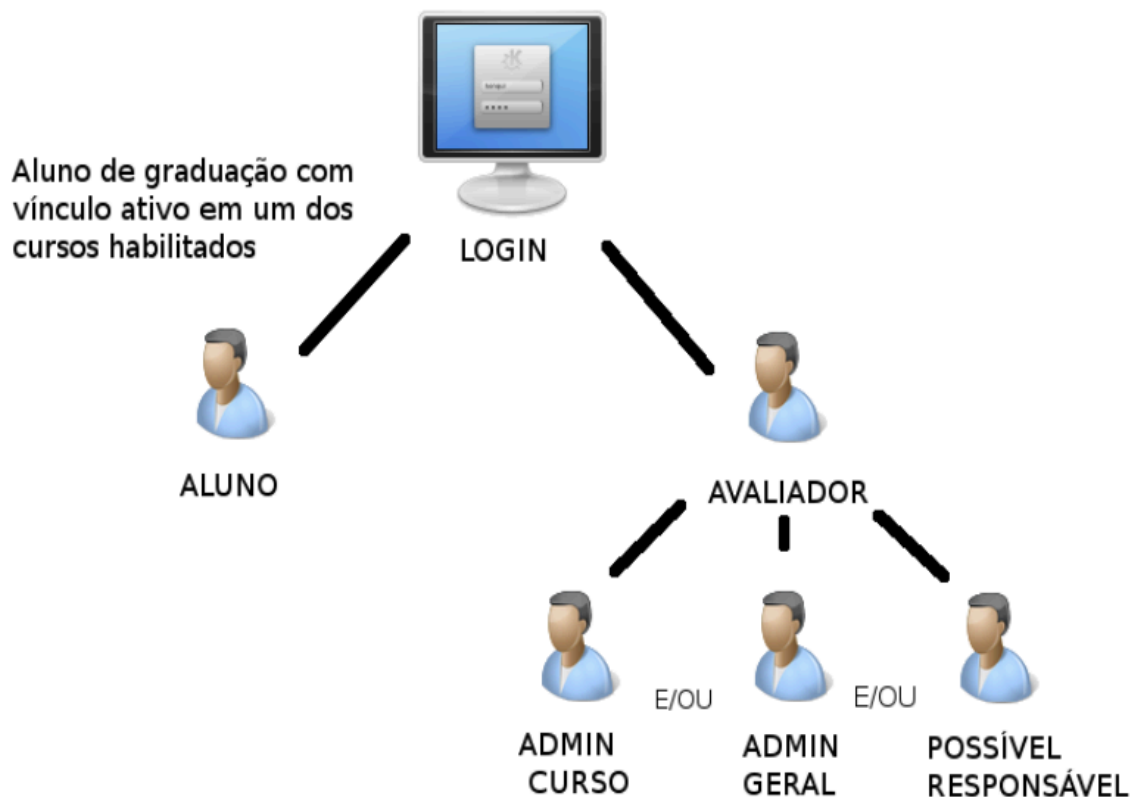


Figura 2: Níveis de acesso. (Botelho e Ugioni, 2015)

2.1.1 Papel aluno

O papel do aluno no Sistema de Gerenciamento de TCCs é central, sendo o principal beneficiário das funcionalidades do sistema. Os alunos utilizam o sistema para gerenciar integralmente seus projetos de conclusão de curso, desde a fase inicial até a etapa de conclusão, passando por diversas atividades intermediárias importantes para o sucesso de seus trabalhos. O diagrama de casos de uso representa as funcionalidades disponíveis no sistema para o aluno (Figura 3).



Figura 3: Casos de uso Aluno (Botelho e Ugioni, 2015)

2.1.2 Papel avaliador

O papel do avaliador atua como um facilitador e orientador no processo de desenvolvimento e avaliação dos TCCs. Este papel é geralmente atribuído a professores e membros do corpo docente que, através do sistema, têm acesso a uma variedade de funcionalidades projetadas para otimizar a gestão dos cursos, disciplinas e o acompanhamento acadêmico dos alunos. Embora o papel de avaliador possa incluir funções específicas como Administrador do Curso, Administrador Geral e Possível Responsável (Botelho e Ugioni, 2015), é importante ressaltar que também pode ser desempenhado exclusivamente na capacidade de avaliação (membros da banca), sem a necessidade de assumir essas subdivisões adicionais.

Importantes melhorias foram introduzidas no papel de avaliador no trabalho de Onghero (2018), que descreve as principais funcionalidades do sistema através de um diagrama de casos de uso (Figura 4):

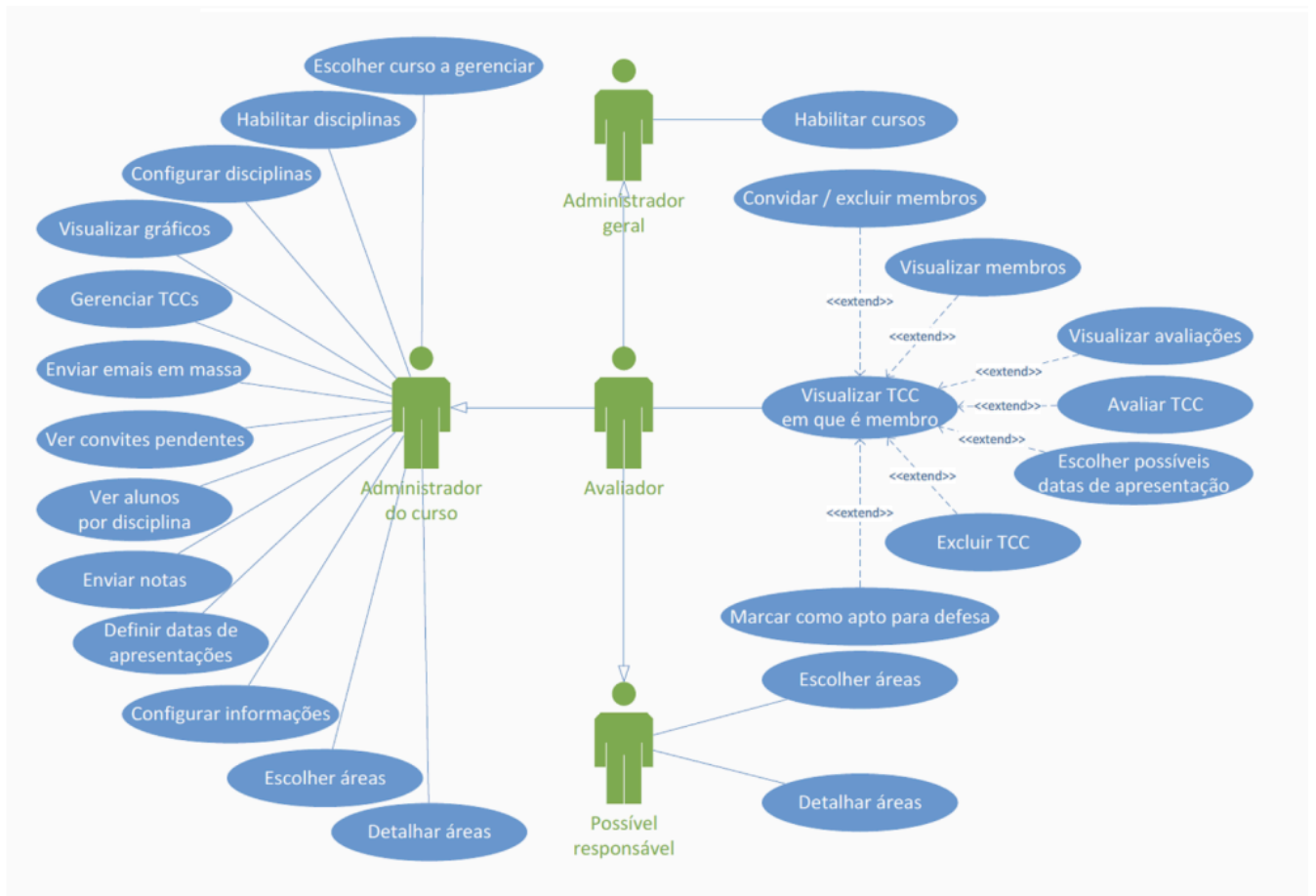


Figura 4: Casos de uso Avaliador (Onghero, 2018)

2.1.3 Módulo de recomendação de temas de TCC

O módulo de recomendação de temas para TCCs, desenvolvido por Leal (2022), aborda uma necessidade crítica enfrentada por alunos nas fases finais de seus cursos de graduação: a escolha de um tema adequado para seus trabalhos de conclusão. Esta ferramenta, integrada ao Sistema de Gestão de TCCs da Universidade Federal de Santa Catarina (UFSC), utiliza *Machine Learning* para sugerir temas baseados nos interesses dos professores e no desempenho acadêmico dos alunos. O sistema analisa os dados de desempenho dos alunos em diversas disciplinas e utiliza essa informação para indicar possíveis áreas de interesse, alinhando-os com os orientadores cujas especialidades correspondem às preferências dos estudantes. A Figura 5 explica como ele funciona:

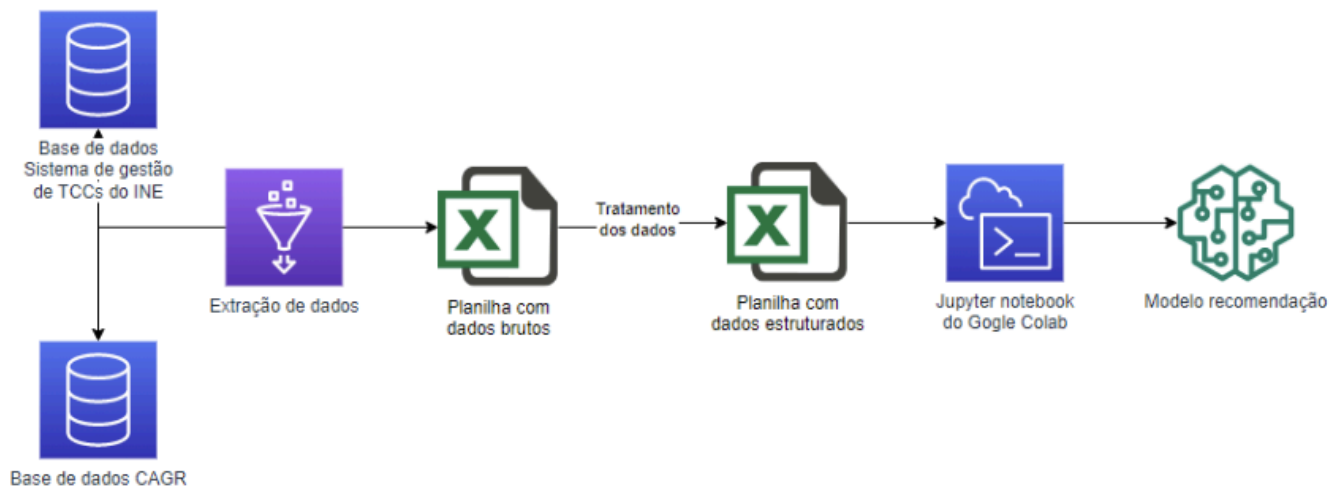


Figura 5: Arquitetura de desenvolvimento do modelo de recomendação (Leal, 2022).

2.2 Containerização

Este trabalho trata da implementação de práticas de Integração e *Deployment* Contínuos (CI/CD) no sistema de TCCs do INE/UFSC. Tanto a integração quanto o *deployment* contínuos podem ser apoiados pelo uso de máquinas virtuais e contêineres.

Vaucher (2015) explica que o papel das máquinas virtuais e dos mecanismos de contêineres é essencialmente o mesmo: fornecer a ilusão de que uma determinada máquina física pode executar múltiplas máquinas. Essas máquinas devem ser isoladas umas das outras e do *host* (máquina física sobre a qual estão sendo executadas). A diferença entre os dois sistemas (máquinas virtuais e *containers*) está em como eles alcançam o isolamento.

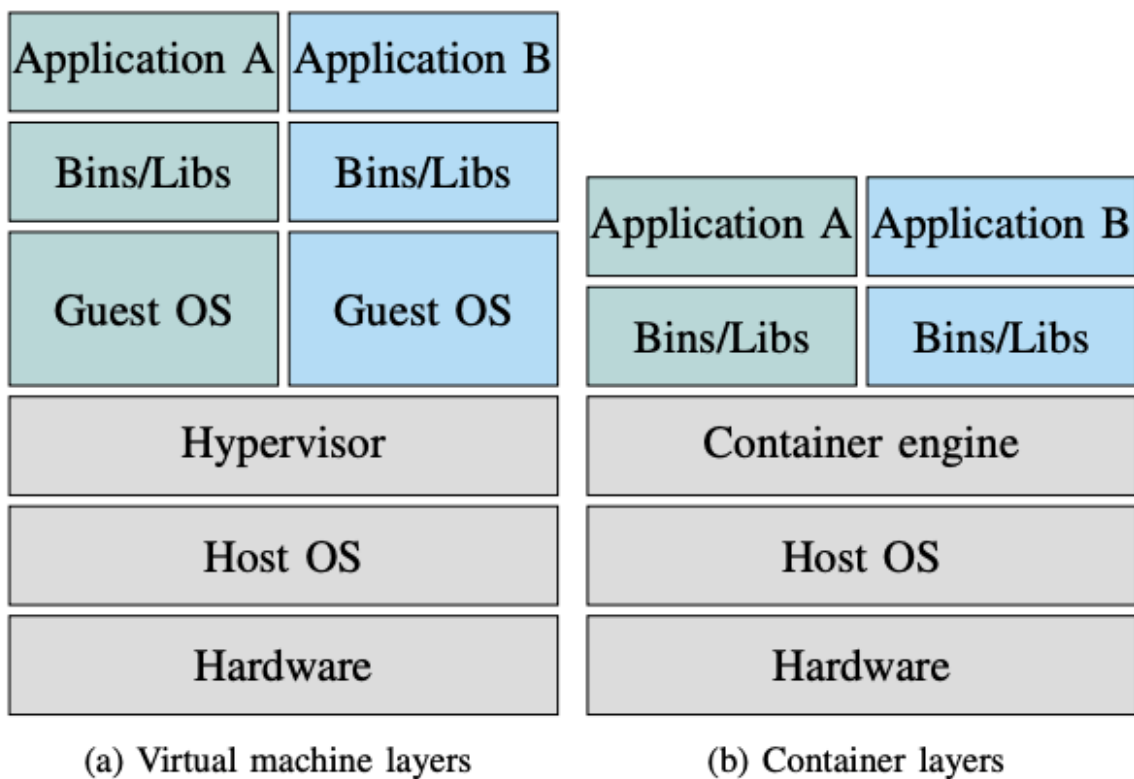


Figura 6: Comparação das camadas envolvidas ao rodar uma aplicação numa máquina virtual ou em um contêiner (Vaucher, 2015)

Como mostrado na Figura 6, máquinas virtuais executam sobre um *hypervisor*, um *software* que permite a criação e gerenciamento de máquinas virtuais isoladas no mesmo *hardware* físico, enquanto contêineres executam no sistema operacional do *host* por meio de um motor (Vaucher, 2015). Resumidamente, pode-se definir um contêiner como sendo uma versão mais leve de uma máquina virtual (Merkel, 2014). Além disso, a prática de containerização possui diversas outras vantagens:

- **Portabilidade:** Aplicações construídas dentro de contêineres são extremamente portáteis, pois esses tipos de pacotes são movidos como uma única unidade e esse movimento não afeta o desempenho ou o próprio *container* de forma alguma (Vase, 2015).
- **Isolamento:** A containerização permite um fácil gerenciamento das dependências para cada aplicação. As bibliotecas específicas, pacotes e configurações exigidas por uma aplicação são agrupadas dentro do contêiner, reduzindo as chances de

conflitos de versão ou problemas de dependência que possam ocorrer (Matthias e Kane 2015).

- **Eficiência de recursos:** Os contêineres são leves e compartilham o *kernel* do sistema operacional do *host*, o que significa que eles consomem menos recursos em comparação com as máquinas virtuais tradicionais. Essa eficiência de recursos permite que as organizações maximizem a utilização do *hardware* e executem mais contêineres na mesma infraestrutura (Syrjämäki, 2023).

2.2.1 *Docker*

A ferramenta *Docker* permite que os desenvolvedores empacotem, distribuam e executem aplicativos em contêineres leves e isolados. O *Docker* levou a virtualização no nível do sistema operacional um passo adiante em comparação com as tecnologias já existentes, como o *Linux Containers* (LXC), oferecendo uma plataforma fácil de usar que automatiza a implantação de aplicativos em contêineres (Turnbull 2014).

O *Docker Engine* é a tecnologia subjacente para criar e colocar aplicativos em *containers*. Ele usa uma arquitetura cliente-servidor e inclui um servidor que abriga o processo *daemon* chamado “*dockerd*”, juntamente com uma API REST e uma interface de linha de comando (CLI) fácil de usar chamada “*docker*” (Docker Inc. 2023). A organização dessa arquitetura é representada na Figura 7.

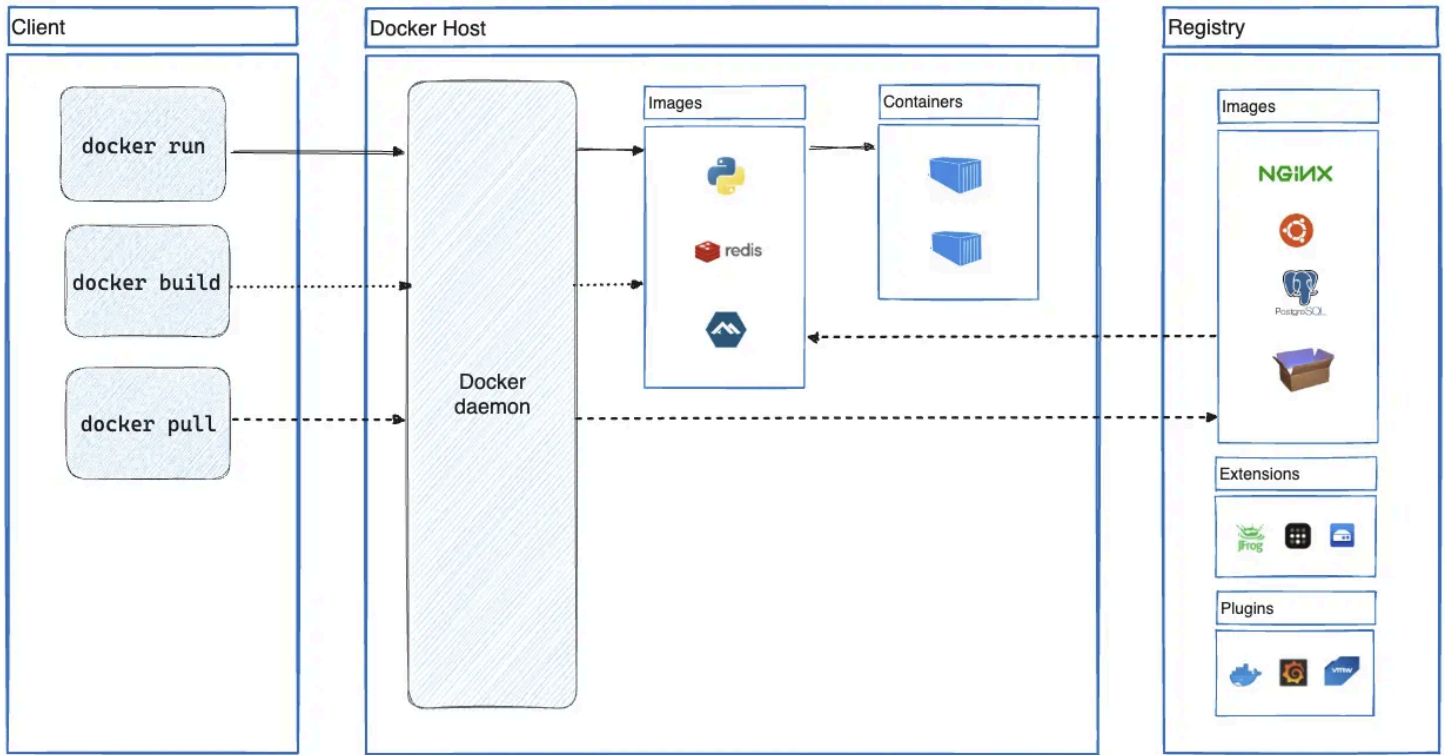


Figura 7: Arquitetura do *Docker* (Docker Inc, 2023)

Os contêineres do *Docker* são baseados em imagens do *Docker*. As imagens em si não são executadas, mas é possível criar e executar contêineres a partir de uma imagem do *Docker*. As imagens do *Docker* contêm instruções para as bibliotecas, dependências e configurações necessárias para o contêiner. As imagens são construídas camada por camada sobre uma imagem de base, que geralmente é um sistema operacional baseado em Linux (Docker Inc. 2023).

As imagens do *Docker* são criadas usando instruções fornecidas em um arquivo chamado *Dockerfile*. Esse arquivo contém todas as instruções para compor o contêiner, e esses contêineres são limitados aos recursos definidos na imagem. O *Docker* cria camadas na ordem especificada no *Dockerfile*, com cada novo comando gerando uma nova camada. Depois que uma imagem é criada, ela pode ser usada como um modelo para iniciar um ou mais contêineres do *Docker*. As imagens do *Docker* são imutáveis, portanto, não podem ser alteradas depois de criadas. Se for necessário fazer alterações, será necessário criar uma nova imagem (Docker Inc. 2023).

2.2.1.1 Serviços e volumes

No contexto do *Docker*, os serviços referem-se a aplicações ou processos que são executados em contêineres isolados. Cada serviço pode ser descrito em um arquivo *docker-compose.yml* (Docker Inc. 2023), facilitando a orquestração de múltiplos contêineres que trabalham juntos como uma aplicação distribuída. Esses serviços são configurados a partir de imagens do *Docker*, que podem ser personalizadas para atender às necessidades específicas de cada serviço.

Já os volumes são fundamentais no *Docker* para o gerenciamento de dados. Eles são usados para persistir dados gerados e usados pelos *containers*, permitindo que dados sobrevivam além do ciclo de vida de um contêiner individual. Os volumes são gerenciados pelo *Docker* e podem ser montados em contêineres, proporcionando um sistema de armazenamento que é separado dos *containers* e não depende da existência de um contêiner específico para manter os dados. Isso é essencial para aplicações que necessitam de persistência de dados, como bancos de dados ou sistemas que armazenam estado do usuário.

O *Docker* permite a configuração de volumes no *Dockerfile* ou, de forma mais flexível, através do *Docker Compose* (Docker Inc. 2023). Ao definir um volume, o usuário pode especificar opções como o caminho no *host* e o caminho de montagem dentro do contêiner, além de configurações de permissões de acesso. Este recurso é crucial para a gestão de dados em aplicações *Docker*, pois garante que dados importantes não sejam perdidos quando um contêiner é parado ou reconstruído.

2.3 Integração e *Deployment* contínuos

A Integração Contínua (CI) é uma prática de desenvolvimento que visa à construção contínua de novo *software* (Duvall, 2010). Quando os desenvolvedores enviam mudanças para um repositório compartilhado, a CI exige compilar, testar e garantir a qualidade do produto de *software* modificado, permitindo que os

desenvolvedores saibam imediatamente como suas mudanças se encaixam no *software* em desenvolvimento. Uma das características mais promissoras da CI é a possibilidade de facilitar a identificação de *bugs*, bem como a melhoria da qualidade do código-fonte (Duvall; Matyas e Glover, 2007).

O *Deployment* Contínuo (CD) é um conjunto de práticas criadas para otimizar o processo de levar as alterações do código para o ambiente de produção ou de testes (Ska e Janini, 2019). A Figura 8 representa as principais etapas de um processo de CD.

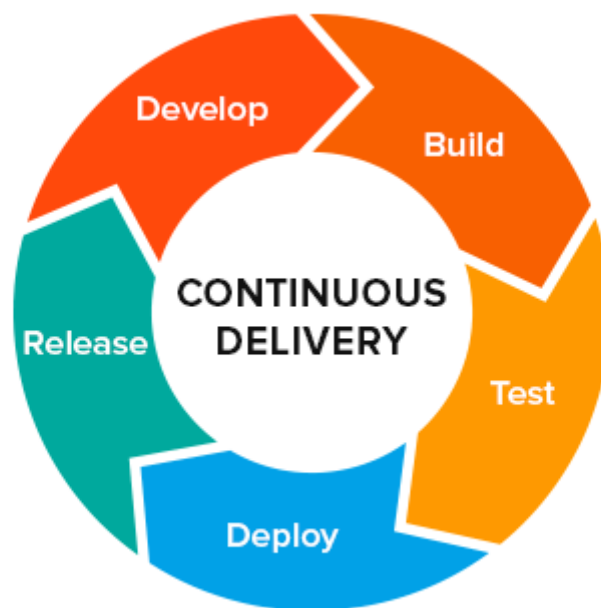


Figura 8: Deployment Contínuo (CD)
(Fonte: <https://developerexperience.io/articles/continuous-delivery>)

O objetivo do *Deployment* Contínuo é encontrar maneiras de fornecer *software* valioso e de alta qualidade de maneira eficiente, rápida e confiável. O *Deployment* Contínuo está diretamente relacionado com a velocidade do mercado e com a rapidez do lançamento, mais rápido do que a concorrência. Essa velocidade de mercado significa um ciclo de *feedback* mais curto e um tempo mais rápido para obter valor. Com um ciclo de *feedback* mais curto, falha-se mais rápido, corrige-se mais rápido, ajusta-se mais rápido e têm-se sucesso mais rápido (Ska e Janini, 2019)

2.3.1 Pipeline de CI/CD

Um *pipeline* CI/CD refere-se ao processo amplamente automatizado de integrar mudanças confirmadas no código, testá-las e, então, mover o código da etapa de confirmação para a etapa de produção (Paule, 2018). *Commits* frequentes, porém pequenos, são encorajados (Fowler, 2006; Glover; Duvall e Matyas, 2007). De fato, uma análise em larga escala conduzida por Zhao et al. (2017) determinou que são feitos, em média, cerca de 21 *commits* por dia para 575 projetos de código aberto. Realizar testes automatizados em cada mudança de código possibilita um *feedback* mais precoce sobre *bugs*, uma vez que os erros podem ser rastreados até suas fontes mais rapidamente. (Humble e Farley, 2010). Pode-se perceber um exemplo disso na Figura 9, obtido a partir da ferramenta GitLab⁶, onde é possível rastrear exatamente qual *commit* falhou.



Figura 9: Lista de pipelines executados

Humble et al. (2010) detalha também que um *pipeline* de CI/CD é composto por múltiplas etapas, iniciando tipicamente com a geração da *build*. Esta é seguida por diversos estágios que incluem testes e *deployments*. A Figura 10 ilustra um exemplo de *pipeline* executado no GitLab, demonstrando seus diferentes passos:

⁶ <https://about.gitlab.com/>

passed Pipeline #8823 triggered 6 months ago by Jean Carlo Rossa Hauck

🕒 5 jobs for dev in 32 minutes and 59 seconds (queued for 2 seconds)

🔗 37ebf638

🔗 1 related merge request: !22 Merge Curso Apps Inteligentes para Todos

Pipeline Needs Jobs 5 Tests 0

Build	Test	Stop_test
✓ build backend	✓ start test server	⚙ stop test server
✓ build database		
✓ build frontend		

Figura 10: Etapas de um pipeline

2.3.2 GitLab

GitLab, uma plataforma integrada de DevOps, oferece uma solução abrangente para o gerenciamento de todo o ciclo de vida do desenvolvimento de *software*, desde a concepção inicial até a entrega final e monitoramento (GitLab Inc., 2023). Diferencia-se de outras soluções como o *Jenkins* por ser uma aplicação única para todo o processo de CI/CD e versionamento do código, facilitando a colaboração entre as equipes de desenvolvimento e operação, ao mesmo tempo em que aumenta a eficiência e reduz os tempos de ciclo de desenvolvimento. Com o GitLab as organizações podem simplificar seus fluxos de trabalho de integração contínua (CI) e entrega contínua (CD), promovendo uma cultura de automação que abraça as práticas de DevOps. Além disso, o GitLab oferece recursos como revisão de código, rastreamento de problemas e *wikis* (Figura 11), tornando-se uma ferramenta essencial para equipes que buscam melhorar a qualidade de seus *softwares* e acelerar o tempo de entrega ao mercado (GitLab Inc., 2023).

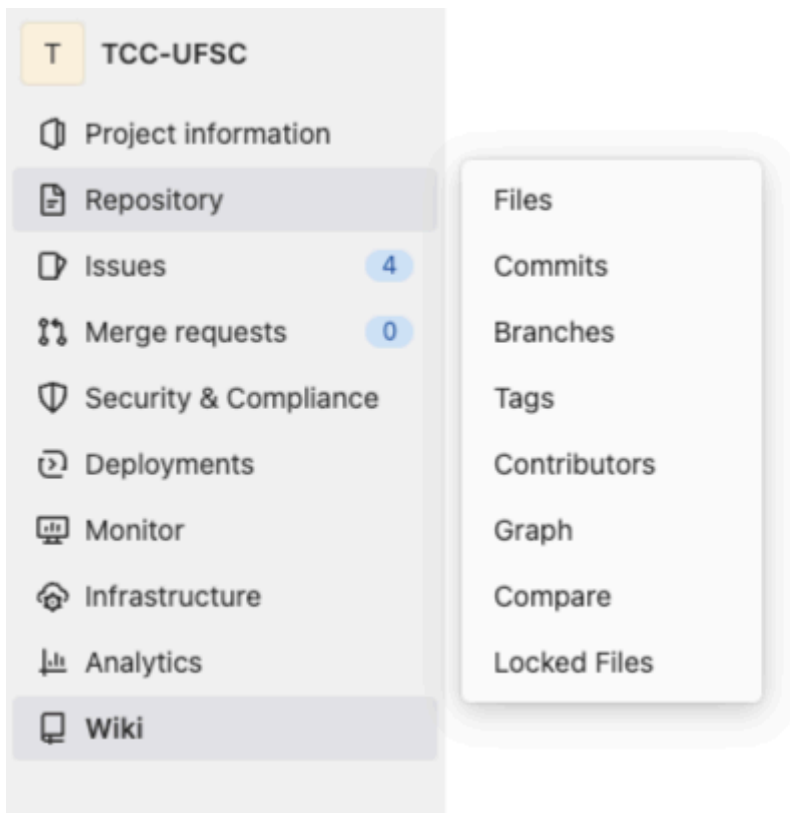


Figura 11: Funcionalidades do GitLab

3 Proposta de Solução

Neste capítulo é apresentada a proposta de solução para a necessidade de implementação de CI/CD para o sistema de gestão de TCCs do INE/UFSC. Inicialmente são levantadas a situação atual de configuração e *deploy* do sistema de TCCs. Na sequência são identificadas as necessidades e restrições de CI/CD para suporte ao desenvolvimento do sistema de gestão de TCCs, por fim são apresentados os métodos de containerização propostos para a solução.

3.1 Análise da situação atual

O desenvolvimento do Sistema de TCCs atualmente enfrenta desafios significativos devido à falta de um processo de CI/CD. As atualizações do sistema em produção são realizadas manualmente, o que aumenta o risco de erros e dificulta a manutenção e a evolução contínua do *software*. Além disso, o ambiente de desenvolvimento varia entre os desenvolvedores, o que pode levar a inconsistências e problemas de compatibilidade. A documentação é escassa, sendo que o guia para configuração do ambiente local, o trabalho de Onghero (2018), tem mais de 6 anos.

Na sequência são apresentados alguns aspectos relevantes do ambiente, estrutura de arquivos, e configuração atuais de desenvolvimento do sistema.

3.1.1 Estrutura de arquivos

A estrutura de arquivos do sistema é organizada da seguinte forma:

- `src/main/java`: Contém os arquivos de código fonte Java
- `src/test/java`: Diretório destinado aos testes unitários e de integração
- `src/test/webapp`: Diretório com o código fonte do sistema de TCCs.
- `pom.xml`: Arquivo de configuração do Maven⁷ que gerencia dependências e *plugins*.

⁷ <https://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html>

- `dependencias`: Diretório especial que contém as dependências do tipo "system" necessárias para o projeto.
- `sql`: Diretório com *scripts* SQL para atualizações manuais no banco de dados.
- `recomendacao`: Diretório contendo o serviço de recomendação elaborado por Leal (2022).

3.1.2 Ambiente de desenvolvimento

O desenvolvimento é realizado em ambientes locais nas máquinas dos desenvolvedores. Cada desenvolvedor tem que configurar seu próprio ambiente, um processo trabalhoso, principalmente porque os passos não estão documentados corretamente, ou são de difícil acesso. Na Figura 12 são mostrados os quatro passos necessários para configurar o ambiente segundo Onghero (2018):

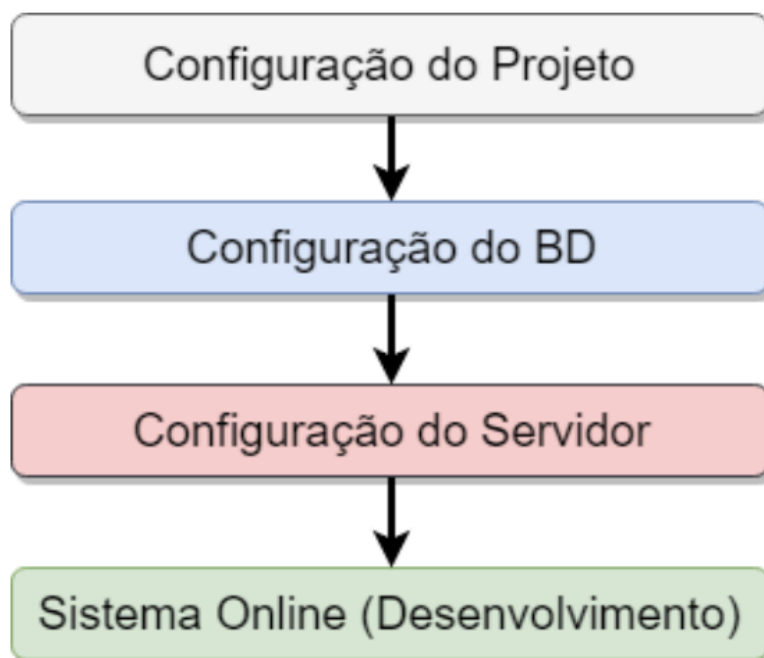


Figura 12: Quatro passos da configuração do ambiente de desenvolvimento (Onghero, 2018)

Na teoria, o passo 1 é simples, porque o próprio IntelliJ IDEA se encarrega de carregar as dependências baseando-se no conteúdo do `pom.xml` e configurar o projeto. Na prática, clonando o repositório da *branch master*, nota-se que o projeto tem problemas graves de instalação (Figura 13).

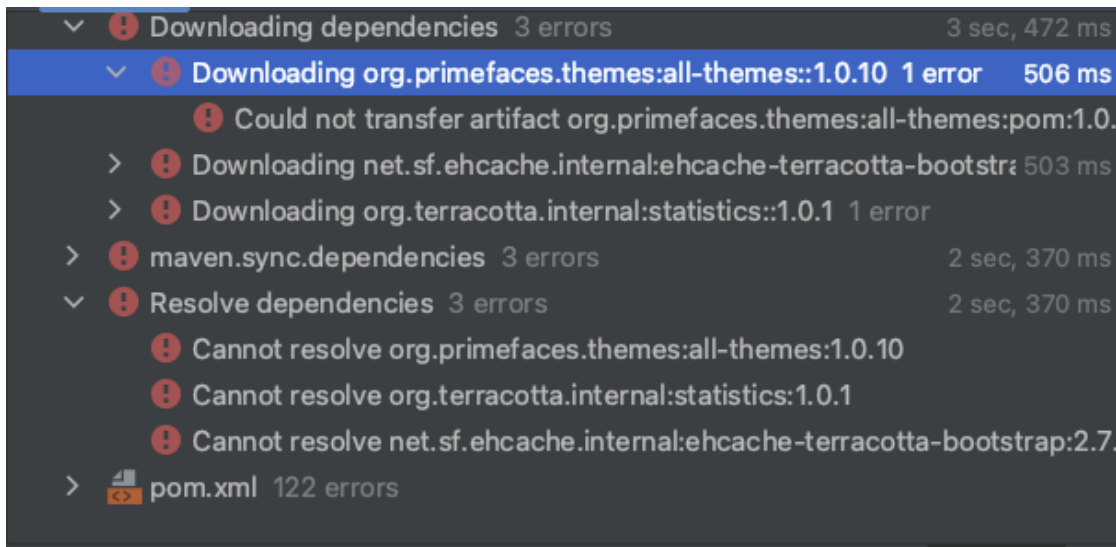


Figura 13: Problemas nas dependências do projeto.

No passo 2 é necessário instalar a versão 5.5.56 do MySQL, lançada há 14 anos (3 de dezembro de 2010), e com o final da sua vida há 5 anos (31 de dezembro de 2018). Devido à idade da versão do MySQL, máquinas mais recentes têm problemas na instalação, o que é o caso do MacBook Pro M1, que falha com o erro `Can't connect to local MySQL server through socket '/tmp/mysql.sock' (2)`.

No passo 3, é necessário baixar o Apache Tomcat,⁸ o servidor utilizado para rodar a aplicação. Necessitando uma configuração específica e detalhada no próprio IntelliJ IDEA (Figura 14) para que a aplicação rode corretamente e consiga se comunicar com o sistema de autenticação centralizada da UFSC (CAS).

⁸ <https://tomcat.apache.org>

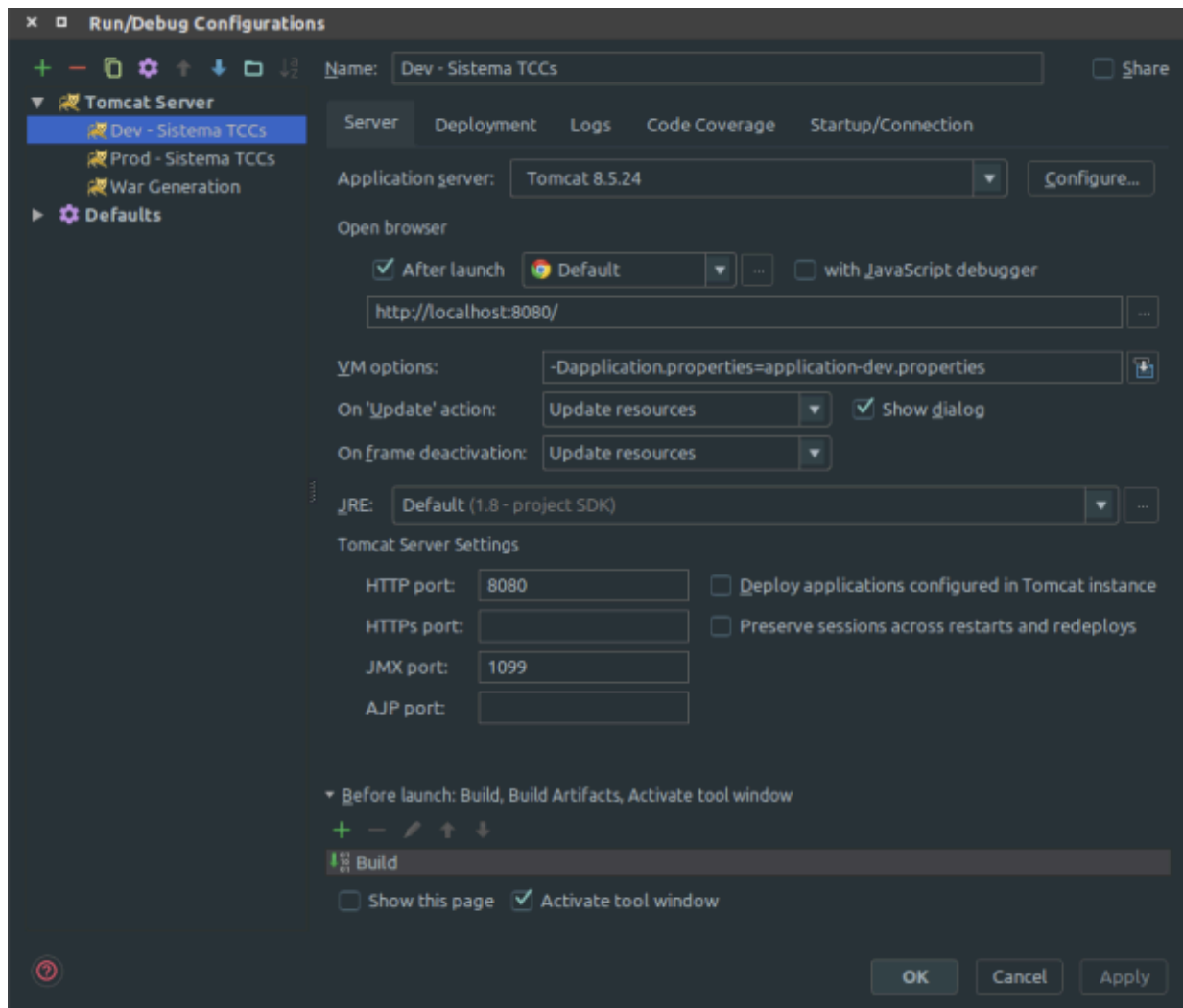


Figura 14: Configuração de execução utilizada no ambiente de desenvolvimento (Onghero 2018)

Caso todos os passos sejam completados corretamente, o ambiente de desenvolvimento estará rodando.

3.1.3 Ambiente de produção

O ambiente de produção consiste em um servidor virtual hospedado pela SETIC⁹ da UFSC, dependendo de processos manuais para *deploy* e manutenção, tornando o sistema suscetível a erros e falhas operacionais. A intervenção manual necessária para atualizações e para reiniciar o servidor em caso de falhas aumenta o risco de interrupções, dificultando a manutenção eficiente e contínua do sistema. Esse processo

⁹ <https://setic.ufsc.br>

de manutenção sobrecarrega os responsáveis, que enfrentam uma pressão constante para assegurar a disponibilidade do sistema.

3.1.4 Levantamento das necessidades de CI/CD

Por meio de reuniões com os responsáveis atuais pelo desenvolvimento e manutenção do sistema de gestão de TCCs, foram levantadas as principais necessidades de CI/CD para o sistema. As necessidades identificadas são apresentadas a seguir (identificadas como RNF - Requisitos Não Funcionais):

- **RNF1 - Facilidade da configuração do ambiente de desenvolvimento:** Automatizar esse processo visa eliminar inconsistências entre ambientes de trabalho e reduz o tempo de preparação necessário para novos desenvolvedores começarem a contribuir para o projeto;
- **RNF2 - Construção e compilação automatizadas:** A automatização do processo de construção e compilação busca garantir que o *software* seja consistentemente construído da mesma maneira em todos os ambientes;
- **RNF3 - Containerização do ambiente:** A containerização tem o intuito de oferecer um ambiente isolado e consistente para desenvolvimento, testes e produção. Além disso, facilita a portabilidade e a escalabilidade do sistema ao mover as aplicações entre diferentes infraestruturas de nuvem ou locais;
- **RNF4 - Ambiente de testes:** Atualmente, a falta de um ambiente dedicado a testes é um gargalo, pois impede a verificação sistemática e automatizada do código antes de alcançar a produção. Implementar um ambiente de testes contínuos dentro do *pipeline* de CI/CD busca garantir a qualidade e a estabilidade do *software*;
- **RNF5 - Integração Contínua (CI):** Com CI se objetiva que cada contribuição de código possa ser automaticamente testada e integrada ao repositório principal, o

que ajuda a detectar e corrigir *bugs* rapidamente, mantendo a qualidade do *software* em alta;

- **RNF6 - *Deployment* Contínuo (CD):** O CD visa permitir que as alterações de código validadas através da CI sejam automaticamente colocadas em produção ou em ambientes de teste, reduzindo significativamente o tempo de entrega de novas funcionalidades e correções

Na sequência, é apresentada a proposta de CI/CD com base nessas necessidades identificadas.

3.2 Proposta de CI/CD para o sistema de TCCs

Nesta seção é apresentada a proposta de solução para os problemas identificados no Sistema de TCCs do INE. A proposta contempla: a containerização do ambiente de execução, a automação de CI/CD e a documentação do processo de DevOps.

3.2.1 Containerização

A containerização é proposta como uma solução robusta para os desafios enfrentados no Sistema de TCCs, oferecendo um método para empacotar e distribuir a aplicação de forma consistente. Utilizando tecnologias de contêiner, como Docker, o ambiente de execução e o código-objeto do sistema podem ser encapsulados juntamente com suas dependências, garantindo que funcione de maneira idêntica em qualquer ambiente de produção, teste ou desenvolvimento.

A Figura 15 apresenta a proposta de containerização do sistema, possuindo alguns volumes e serviços:

- Serviço `app`: opera a aplicação principal, o portal *online* onde alunos e avaliadores interagem com o sistema de TCCs.
- Volume `TCC Files`: é configurado para armazenar os arquivos submetidos pelos usuários do sistema de TCCs, mesmo após o *container* ser reiniciado ou removido.

- Serviço `db`: Gerencia o banco de dados MySQL que armazena informações dos TCCs.
- Volume `Database`: garante que as informações essenciais dos TCCs sejam mantidas seguras e acessíveis, mesmo após o contêiner ser reiniciado ou removido.
- Serviço `recommendation`: opera o serviço de recomendação proposto por Leal (2022).

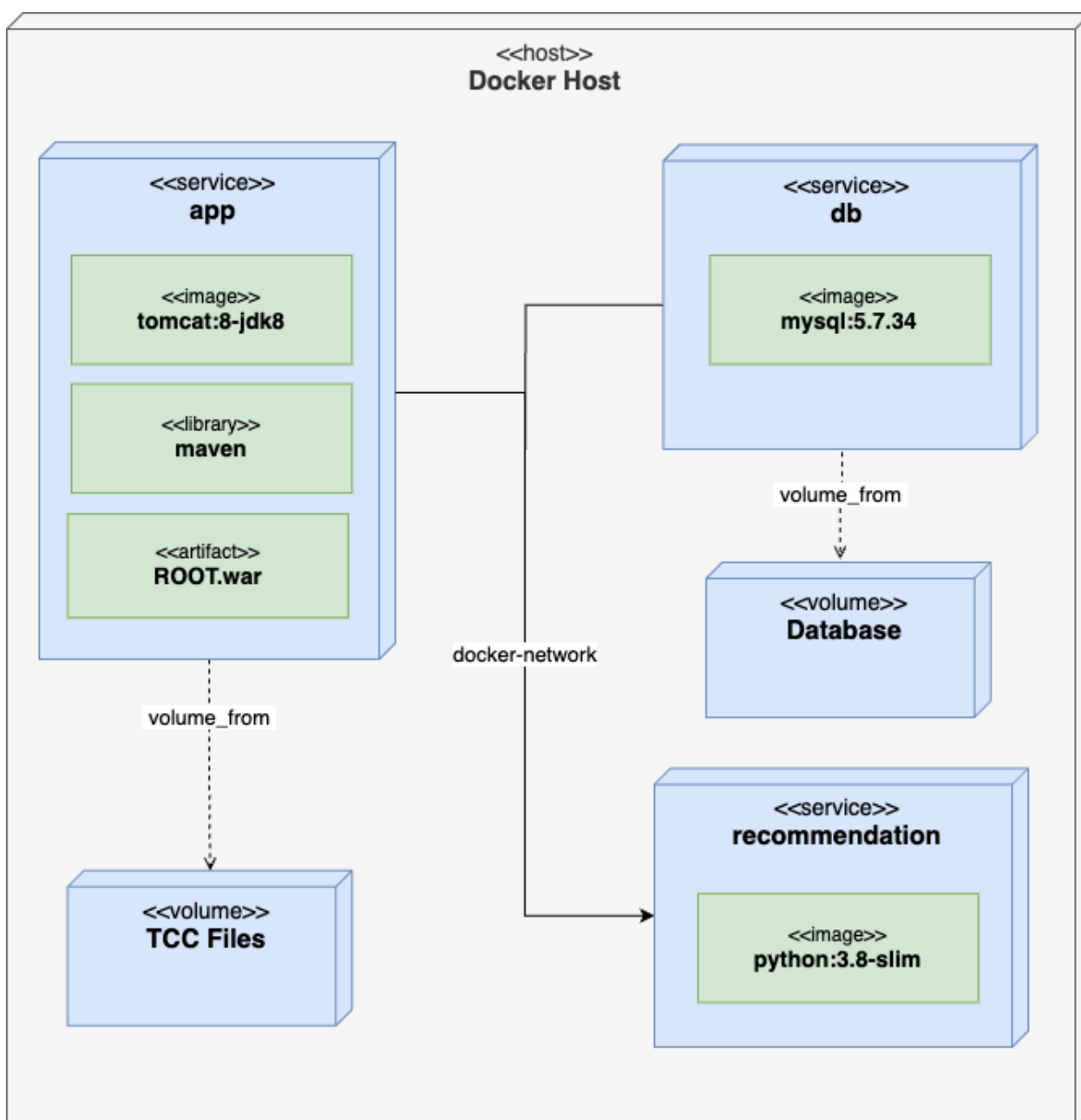


Figura 15: Deployment Diagram (elaborado pelo Autor)

3.2.2 Automação de CI/CD no Gitlab

A automação de CI/CD para o sistema de TCCs será implementada por meio do GitLab, utilizando *pipelines* definidos no arquivo `.gitlab-ci.yml`. Este arquivo configura todas as etapas necessárias, como construção da aplicação, execução de testes e *deployment* em ambientes de teste e produção. A implementação utilizará *runners* específicos, já configurados conforme Andrade (2022), de forma a agilizar a implantação do CI/CD. O objetivo é simplificar o processo de integração e entrega contínuas, permitindo atualizações frequentes e estáveis do sistema, com o mínimo de intervenção manual.

3.2.3 Documentação de DevOps

Para a documentação de DevOps, é proposto reestruturar e expandir a *wiki* no GitLab¹⁰, criando um recurso abrangente que sirva como um manual para o sistema de TCCs. A nova documentação visará a cobrir desde a configuração inicial, passando pela explicação detalhada dos *pipelines* de CI/CD, até orientações sobre a utilização e manutenção do sistema. Incluirá tutoriais passo-a-passo, exemplos de código e melhores práticas de DevOps para ajudar tanto os desenvolvedores quanto os usuários finais a entenderem completamente o sistema e seus componentes. Além disso, a documentação será organizada de forma que novos desenvolvedores possam rapidamente se familiarizar com o projeto e iniciar suas contribuições de maneira eficiente.

3.2.4 Cobertura da Proposta de Solução

O Quadro 1 relaciona as necessidades levantadas anteriormente e como elas são atendidas por uma ou mais propostas de solução.

¹⁰ <https://codigos.ufsc.br/antonio.c.mariani/TCC-UFSC/-/wikis/home>

Necessidade (RNF)	Proposta de solução
RNF1 - Facilidade da configuração do ambiente de desenvolvimento	Containerização; Documentação de DevOps.
RNF2 - Construção e compilação automatizadas	Containerização.
RNF3 - Containerização do ambiente	Containerização.
RNF4 - Ambiente de testes	Containerização; Automação de CI/CD no Gitlab
RNF5 - Integração Contínua (CI)	Containerização; Automação de CI/CD no Gitlab
RNF6 - Deployment Contínuo (CD)	Containerização; Automação de CI/CD no Gitlab

Quadro 1: Necessidades e soluções propostas

Conforme pode ser visto no Quadro 1, todas as necessidades inicialmente levantadas são contempladas pela solução proposta. Destaca-se a Containerização do sistema, que é envolvida no atendimento de todas as necessidades levantadas. A Documentação de DevOps, por sua vez, atende somente a facilidade de configuração, enquanto a Automação de CI/CD cobre a maioria das necessidades técnicas de testes, integração de *deployment* contínuos.

3.3 Implementação inicial da containerização

A implementação inicial da containerização do sistema de gestão de TCCs do Departamento de Informática e Estatística da UFSC envolve uma série de desafios técnicos relacionados ao ambiente legado e à arquitetura da aplicação. O primeiro passo realizado é a definição dos arquivos *Dockerfile* e *docker-compose.yml*, conforme apresentado nas Figuras 16 e 17. Em seguida, detalham-se as dificuldades encontradas e as soluções aplicadas.

```
FROM tomcat:8-jdk8
RUN apt-get update && apt-get install -y \
iputils-ping \
maven \
nodejs \
npm \
git-all \
ant \
cmake \
vim
WORKDIR /tcc-app
RUN mkdir -p /home/tcc/files
COPY . .
EXPOSE 8080
CMD ["mvn", "spring-boot:run"]
```

Figura 16: *Dockerfile* inicial

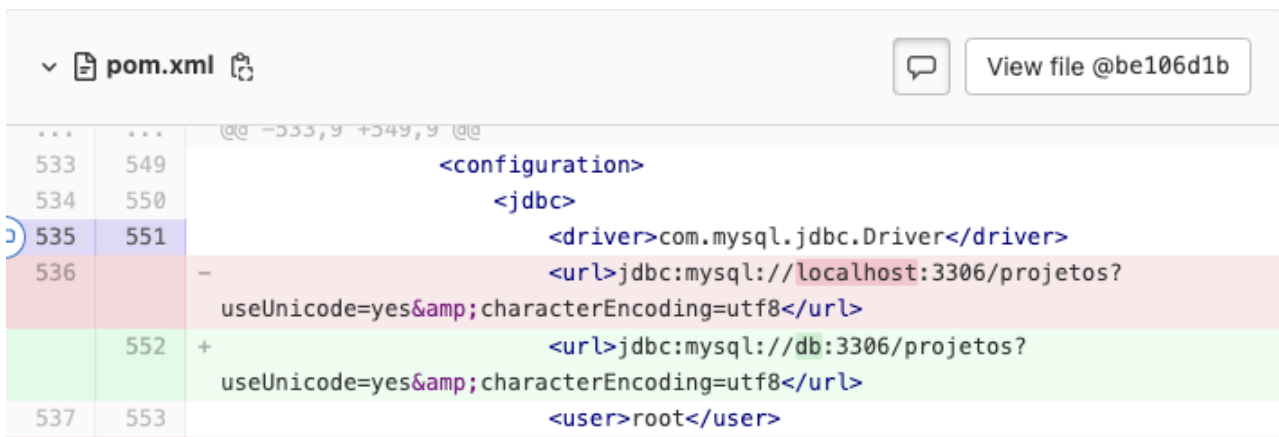
```
services:
  db:
    container_name: "TCC-db"
    platform: linux/x86_64
    image: mysql:5.7.34
    command: --character-set-server=utf8mb4 --max_allowed_packet=128M --innodb_log_file_size=256M
    env_file: ./env
    environment:
      MYSQL_ROOT_PASSWORD: $MYSQLDB_ROOT_PASSWORD
      MYSQL_DATABASE: $MYSQLDB_DATABASE
    volumes:
      - my-db:/var/lib/mysql
    ports:
      - '3306:3306'
  app:
    container_name: "TCC-app"
    depends_on:
      - db
    restart: on-failure
    env_file: ./env
    ports:
      - '8080:8080'
    volumes:
      - ./tcc-files:/home/tcc/files
      - ./tcc-app
    stdin_open: true
    tty: true
volumes:
  my-db:
```

Figura 17: *docker-compose.yml* inicial

3.3.1 Problemas de rede

Inicialmente, enfrentou-se um desafio significativo com a configuração de rede. A aplicação estava originalmente configurada para conectar-se ao banco de dados utilizando *localhost*, o que funcionava em ambientes não containerizados. No entanto, ao mover para contêineres, essa configuração resulta em falhas de conexão, pois *localhost* referenciava o próprio container da aplicação, e não o container do serviço de banco de dados.

A solução foi modificar a *string* de conexão JDBC de *localhost* para *db*, correspondendo ao nome do serviço do banco de dados definido no *docker-compose.yml*, conforme mostra a Figura 18:



```
...      ...      @@ -533,9 +549,9 @@
533      549      <configuration>
534      550      <jdbc>
535      551      <driver>com.mysql.jdbc.Driver</driver>
536      -      <url>jdbc:mysql://localhost:3306/projetos?
        useUnicode=yes&characterEncoding=utf8</url>
552      +      <url>jdbc:mysql://db:3306/projetos?
        useUnicode=yes&characterEncoding=utf8</url>
537      553      <user>root</user>
```

Figura 18: Ajustando o *host* do banco de dados

3.3.2 Adaptação do projeto para uso do Tomcat

Descobriu-se que o projeto não era compatível com *Spring Boot* quando tentativas de usar `mvn spring-boot:run` falharam, resultando em erros como "*Command not found*" ou "*Non-resolvable parent POM*". Isso indicava a necessidade de uma abordagem tradicional usando o Apache Tomcat. Portanto, o *Dockerfile* foi ajustado para instalar o Tomcat e colocar o arquivo *.war* gerado pelo Maven no diretório apropriado (Figura 19).

```
# Não funciona com a versão atual do spring
# CMD ["mvn", "spring-boot:run"]

# Solução: Gera o arquivo war e utiliza o Tomcat
RUN mvn install
RUN cp target/projetostcc.war /usr/local/tomcat/webapps/ROOT.war
RUN catalina.sh run
```

Figura 19: Utilizando o Tomcat

3.3.3 Otimização do ciclo de vida do Maven

A ineficiência do processo de *build* era evidente, com dependências sendo desnecessariamente baixadas em cada nova construção da imagem. Isso resultava em tempo de *build* prolongado, de aproximadamente 18 minutos.

Então, adotou-se o uso do `mvn dependency:go-offline` na fase de *build* da imagem *Docker*. Isso permitiu que todas as dependências necessárias fossem baixadas e armazenadas em cache. Posteriormente, o comando `mvn package` foi executado para compilar o projeto sem reavaliar as dependências (Figura 20). Essa abordagem otimizou significativamente o tempo de *build* (18 minutos para 30 segundos) e reduziu o *overhead* de rede.

```
# Copiando e instalando as dependências - utiliza o mecanismo de cache do Docker
# (Menos se houver mudanças no diretório `dependências` ou no arquivo `pom.xml`)
COPY ./dependências ./dependências
COPY pom.xml .
RUN mvn dependency:go-offline

# Copia o resto da aplicação para dentro do container
COPY . .

# Construindo a aplicação sem a necessidade de reinstalar as dependências
RUN mvn package
RUN cp target/projetostcc.war /usr/local/tomcat/webapps/ROOT.war
RUN catalina.sh run
```

Figura 20: Otimizando a construção da imagem

4 Implementação

Neste capítulo, é apresentada a implementação do processo de Integração e Deployment Contínuos (CI/CD) no Sistema de Trabalhos de Conclusão de Curso (TCCs) do departamento INE da UFSC. A implementação abrange desde a configuração das pipelines de CI/CD até o *deployment* nos ambientes de teste e produção. Além disso, são discutidos aspectos relacionados à criação da documentação de DevOps, garantindo que o processo seja replicável e compreensível para os futuros desenvolvedores.

4.1 Configuração das Pipelines de CI/CD

A configuração das pipelines de CI/CD para o Sistema de Gestão de TCCs foi uma etapa crítica na implementação do processo de integração e *deployment* contínuos. Essa fase envolveu a definição de um fluxo automatizado que garantisse a construção consistente das imagens *Docker* e a implantação segura nos ambientes de teste e produção. A abordagem adotada se baseou em duas fontes principais: o trabalho de Andrade (2022), que serviu como referência para os conceitos de CI/CD, e as instruções específicas fornecidas pela SETIC, que orientaram as adaptações necessárias ao ambiente de infraestrutura da UFSC.

4.1.1 Estruturação das Pipelines

Para iniciar a estruturação das pipelines de CI/CD, foi necessário compreender a configuração pré-existente do GitLab e habilitar recursos adicionais que permitissem o armazenamento de imagens *Docker* no *registry* da UFSC. Seguindo a orientação de Andrade (2022) e as diretrizes da SETIC, o recurso "*Packages & Registries*" no GitLab foi ativado (Figura 21), o que garantiu o acesso às variáveis de ambiente `$CI_REGISTRY_IMAGE` e `$CI_REGISTRY` no arquivo `.gitlab-ci.yml`. Essas variáveis são essenciais para identificar o destino correto das imagens *Docker* no *registry* da SETIC durante os processos de *build* e *push*.

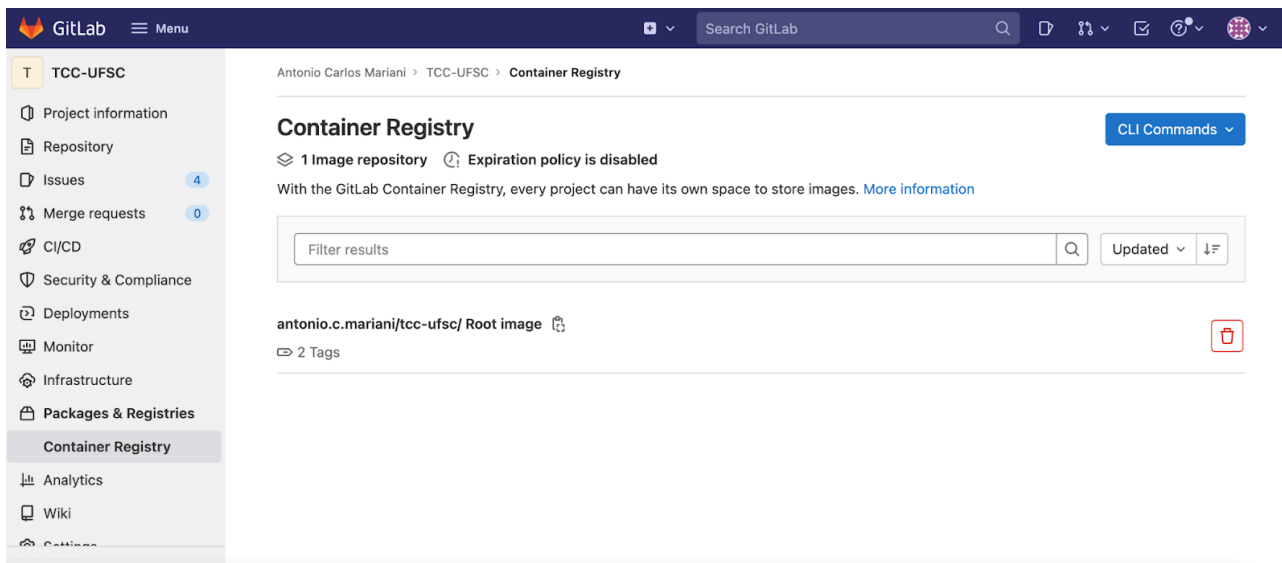


Figura 21: *Package & Registries* habilitado

4.1.2 Integração com o GitLab CI/CD

Com a estruturação do *docker compose* e a configuração dos "*Packages and Registries*" no GitLab concluídas, a próxima etapa envolveu a criação do arquivo *.gitlab-ci.yml*, utilizando um *template* pré-definido pela SETIC.

O pipeline é dividido em duas etapas principais: *build* e *portainer*. A etapa de *build* é responsável pela construção da imagem *Docker* e pelo *push* para o *registry* da SETIC/UFSC, enquanto a etapa de *portainer* é dedicada ao *deployment*, utilizando um *webhook* fornecido pela SETIC para reiniciar o serviço. Na Figura 22 abaixo, o arquivo *.gitlab-ci.yml* é apresentado com a configuração utilizada:

```

image: docker:27-cli

stages:
  - build
  - portainer
  ### BUILD
build:
  stage: build
  only:
    - master
    - main
  script:
    - docker build . -t "${CI_REGISTRY_IMAGE}:${CI_COMMIT_REF_SLUG}" --no-cache
    - docker login $CI_REGISTRY -u $CI_REGISTRY_USER -p $CI_REGISTRY_PASSWORD
    - docker push "${CI_REGISTRY_IMAGE}:${CI_COMMIT_REF_SLUG}"

portainer:
  stage: portainer
  only:
    - master
    - main
  variables:
    GIT_CHECKOUT: "false"
  script:
    - if [[ ! -z "${WEBHOOK_PORTAINER}" ]]; then
      for url in $(echo ${WEBHOOK_PORTAINER} | sed "s/,/ /g"); do wget --post-data '' -O - "$url"; done
    else
      echo "Empty \${WEBHOOK_PORTAINER}" ; exit 1;
    fi

```

Figura 22: Arquivo .gitlab-ci.yml

Para viabilizar a execução do pipeline, a variável de ambiente **WEBHOOK_PORTAINER** foi configurada no GitLab com o suporte da SETIC, que também forneceu a URL necessária para o *webhook* de reinício do serviço no *Portainer*. Essa integração permitiu automatizar o *deployment*, garantindo a continuidade do serviço sem a necessidade de intervenção manual.

4.2 Implantação do CI/CD

A implantação do CI/CD para o Sistema de Gestão de TCCs é um processo essencial para assegurar a continuidade e a eficiência do desenvolvimento e *deploy* do sistema. Essa fase inclui a configuração do ambiente de testes (homologação) e as preparações para futura migração ao ambiente de produção. A seguir são descritas as principais decisões e abordagens tomadas para o tratamento do ambiente de testes

(homologação), gerenciamento de arquivos estáticos e a configuração do banco de dados.

4.2.1 Servidor de homologação

O servidor de homologação¹¹ foi implantado pela equipe da SETIC utilizando um `docker-compose` especialmente configurado para o ambiente de homologação (Figura 23). Este arquivo define as variáveis e volumes necessários para o correto funcionamento do sistema e inclui detalhes específicos para o ambiente de homologação. A implementação foi gerenciada através do *Portainer*¹², uma ferramenta amplamente utilizada para a administração de ambientes *Docker*.

```
3  version: "3.8"
4
5  x-app-base:
6    &app-base
7    restart: on-failure
8    environment:
9      - MYSQLDB_HOST=<host>
10     - MYSQLDB_USER=<user>
11     - MYSQLDB_DATABASE=<database>
12     - MYSQLDB_ROOT_PASSWORD=<senha>
13     - JAVA_OPTS="-Dspring.profiles.active=prod"
14    ports:
15     - 8443:8080
16    volumes:
17     - hmg:/home/tcc/files
18
19    configs:
20     tccine:
21       external: true
22
23    services:
24     homologacao:
25       <<: *app-base
26       image: registry.codigos.ufsc.br/antonio.c.mariani/tcc-ufsc:master
27
28    volumes:
29     hmg:
```

Figura 23: docker-compose.yml de homologação

Após configurar o servidor de homologação, a SETIC forneceu a URL do webhook do *Portainer*. Essa URL foi integrada ao pipeline de CI/CD do projeto, permitindo que o sistema automaticamente reiniciasse o serviço sempre que um novo *deploy* fosse

¹¹ <https://tccdev.sites.ufsc.br/>

¹² <https://www.portainer.io/>

realizado (Figura 24). Essa abordagem garante um fluxo contínuo e eficiente de atualização, minimizando a necessidade de intervenções manuais no ambiente de homologação.

Commit 5e4bb80d authored 2 days ago by mendespedro

testing if ci/cd is working

The screenshot shows a GitHub Actions pipeline run. At the top, it indicates the parent commit is 23e3c711 and the branch is master. Below this, it states 'No related merge requests found'. A green checkmark icon indicates that the pipeline #13198 passed with two stages in 59 seconds. Below the pipeline summary, there are tabs for 'Changes' (1) and 'Pipeline' (1). The 'Pipeline' tab is active, showing a table with columns for Status, Pipeline, Triggerer, and Stages. The pipeline 'testing if ci/cd is working' is listed with a green checkmark status, a duration of 00:00:59, and a triggerer icon. The stages column shows two green checkmarks.

Status	Pipeline	Triggerer	Stages
	testing if ci/cd is working #13198 master 5e4bb80d		

Figura 24: CI/CD funcionando

4.2.2 Servidor de produção

Por uma questão de cronograma e segurança, a migração para o DNS de produção não será realizada no escopo deste trabalho. O sistema de TCCs está em uso contínuo, especialmente durante o final do semestre, quando há um aumento significativo de acessos devido ao processo de entrega e avaliação dos trabalhos. Dessa forma, qualquer alteração que pudesse impactar o uso do sistema foi cuidadosamente evitada.

Após testes extensivos no servidor de testes, a migração para o DNS de produção será realizada em janeiro/2025, período de menor uso do sistema. Nesse momento, o servidor de testes assumirá o papel de servidor de produção, já configurado com o pipeline de CI/CD completo, permitindo uma transição suave e sem interrupções para os usuários.

4.2.3 Arquivos estáticos

Para o armazenamento e o gerenciamento de arquivos estáticos foi configurado um volume no *docker-compose*, mapeado para o diretório `/home/tcc/files` dentro do *container*. Esse diretório, fora do *container*, foi configurado pela equipe da SETIC como um compartilhamento NFS (*Network File System*), permitindo que os arquivos estáticos sejam persistidos de forma centralizada e acessível.

Em uma configuração de NFS, um diretório é criado no servidor de arquivos, exportado e montado nos hosts que precisam acessar o recurso. Esse processo envolve a definição de permissões de acesso e a configuração de comandos como *mount* para estabelecer o ponto de montagem no sistema *host*. No contexto do sistema de TCCs essa configuração realizada pela SETIC assegura que os arquivos estáticos fiquem centralizados e persistam independentemente do ciclo de vida dos containers.

4.2.4 Banco de dados

O sistema utiliza uma instância MySQL monolítica gerida pela SETIC, que centraliza os bancos de dados de vários projetos da UFSC. Para o Sistema de TCCs foi criada uma base de dados específica dentro dessa instância, com credenciais de acesso (usuário e senha) fornecidas pela SETIC.

Após a configuração inicial do banco de dados, foi feito um *restore* de um arquivo `.sql` proveniente do ambiente de produção, com o objetivo de replicar dados reais e oferecer um ambiente de homologação mais fiel ao sistema em uso. Essa abordagem permite realizar validações com dados reais, garantindo que o sistema funcione conforme esperado antes da migração definitiva para o ambiente de produção.

4.2.5 HTTPS e Proxy Reverso

Para garantir a segurança no acesso ao Sistema de Gestão de TCCs em ambiente de produção, a comunicação com o servidor utiliza HTTPS por meio de uma configuração

de proxy reverso gerenciada pela SETIC. Embora o sistema rode no Tomcat na porta 80 com HTTP puro, a SETIC configura um proxy reverso que intercepta as conexões HTTPS e encaminha as requisições para o servidor interno em HTTP. É possível observar um indicativo disso no mapeamento das portas no arquivo *docker-compose* de homologação (Figura 25).

```
x-app-base:
  &app-base
  restart: on-failure
  environment:
    - MYSQLDB_HOST=<host>
    - MYSQLDB_USER=<user>
    - MYSQLDB_DATABASE=<database>
    - MYSQLDB_ROOT_PASSWORD=<senha>
    - JAVA_OPTS="-Dspring.profiles.active=prod"
  ports:
    - 8443:8080
  volumes:
    - hmg:/home/tcc/files
```

Figura 25: Mapeamento da proxy reversa

Essa abordagem foi necessária para alinhar o sistema à infraestrutura da UFSC e permitir que o sistema mantenha a segurança esperada sem a necessidade de gerenciamento de certificados no escopo de cada aplicação.

4.3 Documentação de DevOps

A Documentação de DevOps foi criada com o objetivo de garantir que o processo de configuração e implantação do Sistema de Gestão de TCCs seja replicável e de fácil acesso para desenvolvedores futuros. A documentação detalhada é essencial para que novos integrantes da equipe compreendam o funcionamento do pipeline de CI/CD e possam reproduzir as configurações em diferentes ambientes, tanto de desenvolvimento quanto de produção. Nesta seção, são descritos os passos realizados para a criação e

estruturação da Wiki no GitLab¹³ do projeto, bem como as melhorias implementadas no arquivo README.md¹⁴.

4.3.1 Atualização da Wiki

A Wiki do projeto (Figura 26) foi aprimorada para abranger instruções práticas para desenvolvedores que desejam rodar o projeto em suas máquinas locais e também para orientá-los sobre a execução em ambientes de homologação e produção. Além dessas instruções, foram adicionadas informações sobre a estrutura técnica do projeto e o pipeline de DevOps, oferecendo um repositório de conhecimento mais consolidado e útil tanto para novos desenvolvedores quanto para membros experientes da equipe.

Guia de desenvolvimento para o sistema de TCCs do INE/CTC/UFSC

Ao longo deste guia, são abordados aspectos técnicos e operacionais relacionados à configuração do ambiente de desenvolvimento, documentação técnica do sistema e práticas recomendadas para a manutenção dos servidores de homologação e produção. Este material busca consolidar o conhecimento acumulado durante o desenvolvimento do sistema, servindo como uma referência valiosa para desenvolvedores atuais e futuros, além de contribuir para a sustentabilidade e evolução contínua do sistema.

Ambiente de desenvolvimento

O link abaixo leva a um README detalhado com as instruções necessárias para configurar o ambiente de desenvolvimento. Além disso, o documento inclui uma seção dedicada a problemas comuns e suas possíveis soluções:

- [Preparação do Ambiente](#)

Guia de devops

A atualização mais recente do sistema incluiu a containerização completa do ambiente, representando um avanço significativo na modernização da infraestrutura. Este trabalho consolidou uma grande quantidade de informações técnicas e práticas, servindo como um Guia de DevOps para futuras manutenções e evoluções do sistema. O guia completo pode ser acessado através do seguinte link:

- [Guia de DevOps](#)

Trabalhos passados

- [Monografia da elaboração da primeira versão do sistema](#)
- [Monografia das melhorias de usabilidade para Aluno](#)
- [Dicionário de Dados](#)

Manutenção do servidor de homologação

O servidor de homologação é mantido pela SETIC.

Manutenção do servidor de produção

- [Conexão com o Servidor](#)
- [Reiniciar Tomcat](#)

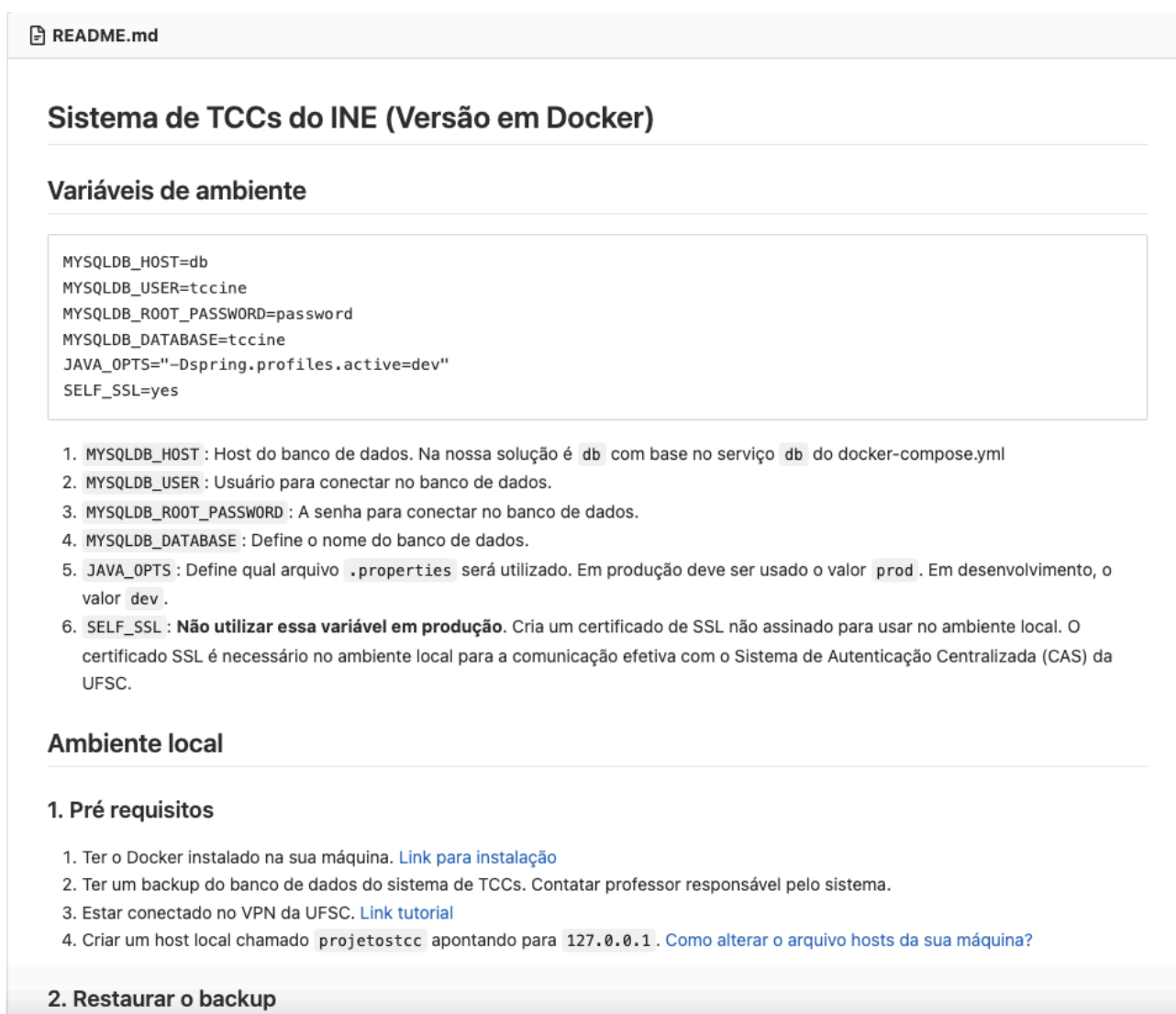
Figura 26: Wiki do projeto

¹³ <https://codigos.ufsc.br/antonio.c.mariani/TCC-UFSC/-/wikis/home>

¹⁴ <https://codigos.ufsc.br/antonio.c.mariani/TCC-UFSC/-/blob/master/README.md>

4.3.2 Atualização do README.md

O arquivo README.md (Figura 27) também foi atualizado para facilitar o *onboarding* de novos desenvolvedores. Agora ele oferece uma introdução simplificada sobre o projeto e orientações claras sobre os passos essenciais para instalar e configurar o sistema localmente. Essa atualização torna o README.md uma referência prática para desenvolvedores que estão começando a trabalhar no sistema, ajudando-os a se familiarizarem rapidamente com as configurações e as dependências do projeto.



README.md

Sistema de TCCs do INE (Versão em Docker)

Variáveis de ambiente

```
MYSQLDB_HOST=db
MYSQLDB_USER=tccine
MYSQLDB_ROOT_PASSWORD=password
MYSQLDB_DATABASE=tccine
JAVA_OPTS="--Dspring.profiles.active=dev"
SELF_SSL=yes
```

1. `MYSQLDB_HOST`: Host do banco de dados. Na nossa solução é `db` com base no serviço `db` do `docker-compose.yml`
2. `MYSQLDB_USER`: Usuário para conectar no banco de dados.
3. `MYSQLDB_ROOT_PASSWORD`: A senha para conectar no banco de dados.
4. `MYSQLDB_DATABASE`: Define o nome do banco de dados.
5. `JAVA_OPTS`: Define qual arquivo `.properties` será utilizado. Em produção deve ser usado o valor `prod`. Em desenvolvimento, o valor `dev`.
6. `SELF_SSL`: **Não utilizar essa variável em produção.** Cria um certificado de SSL não assinado para usar no ambiente local. O certificado SSL é necessário no ambiente local para a comunicação efetiva com o Sistema de Autenticação Centralizada (CAS) da UFSC.

Ambiente local

1. Pré requisitos

1. Ter o Docker instalado na sua máquina. [Link para instalação](#)
2. Ter um backup do banco de dados do sistema de TCCs. Contatar professor responsável pelo sistema.
3. Estar conectado no VPN da UFSC. [Link tutorial](#)
4. Criar um host local chamado `projetostcc` apontando para `127.0.0.1`. [Como alterar o arquivo hosts da sua máquina?](#)

2. Restaurar o backup

Figura 27: README do projeto

4.4 Avaliação

Para validar a implementação do processo de CI/CD no Sistema de Gestão de TCCs, foi realizada uma avaliação com base no método de painel de especialistas (*expert panel*), uma abordagem reconhecida em engenharia de *software* para obter *feedback*

técnico e fundamentado de profissionais experientes (Beecham et al. 2005). Essa avaliação foi conduzida por meio de reuniões com cinco especialistas na área, onde foram apresentadas as soluções implementadas, incluindo a containerização, a configuração das pipelines de CI/CDs e a documentação de DevOps.

Devido à confidencialidade do código-fonte e das configurações específicas do sistema TCC-UFSC, o código não pôde ser compartilhado diretamente. Assim, a apresentação focou em descrever as decisões técnicas tomadas. Posteriormente foram apresentadas algumas perguntas ao painel de especialistas.

4.4.1 Planejamento e Coleta de Dados

A técnica de amostragem utilizada foi a de conveniência, selecionando profissionais com conhecimento em *Docker* e CI/CD. Esses especialistas foram escolhidos com base na facilidade de acesso e disponibilidade para participar da reunião de avaliação.

Para caracterizar os especialistas, foi utilizada a Tabela 1 contendo informações sobre o papel desempenhado atualmente, a empresa ou organização em que trabalham e o tempo de experiência na área.

Formação	Papel atual	Experiência
Graduando em Sistemas de informação	CTO	5 anos
Graduado em Sistemas de Informação	Engenheiro de <i>software</i>	5 anos
Graduado em Ciência da Computação	Engenheiro de <i>software</i>	4 anos
Autodidata	Engenheiro de DevOps	3 anos
Graduado em Ciência da Computação	Engenheiro de DevOps	5 anos

Tabela 1: Caracterização dos Especialistas

As perguntas foram elaboradas para avaliar aspectos fundamentais da implementação, incluindo a eficácia do Guia de DevOps, a configuração do pipeline de CI/CD e o processo de containerização. A seguir são apresentadas as perguntas realizadas durante o painel de especialistas:

1. Como você avaliaria a usabilidade do Guia de DevOps? **Alternativas:**
 - a. Eu acho o guia desnecessariamente complexo;
 - b. Eu achei o guia fácil de usar;
 - c. Eu precisei de ajuda de uma pessoa com conhecimentos técnicos para usar o guia;
 - d. Eu acho que as várias partes do guia estão muito bem integradas;
 - e. Eu acho que o guia apresenta muita inconsistência;
 - f. Eu precisei aprender várias coisas novas antes de conseguir usar o guia;
 - g. Eu imagino que as pessoas aprenderão como usar esse guia rapidamente.
2. A ordem em que os passos do Guia foram colocados é adequada? **Alternativas:**
 - a. Sim
 - b. Não
3. Quais os principais pontos positivos do *pipeline* CI/CD e da containerização?
4. Quais os principais pontos negativos do *pipeline* CI/CD e da containerização?
5. Você tem alguma sugestão para melhorar o *pipeline* ou a containerização?

4.4.2 Resultados

As discussões realizadas com os especialistas forneceram uma visão abrangente sobre a implementação atual, destacando pontos fortes como automação e portabilidade, enquanto também identificaram oportunidades claras de melhoria, especialmente na simplificação da configuração inicial e na revisão da dependência do banco de dados durante a *build*. Os resultados podem ser observados na Figura 28:

4.4.2.1 Como você avaliaria a usabilidade do Guia de DevOps?

Como você avaliaria a usabilidade do Guia de DevOps?

5 responses



Figura 28: Resultados da pergunta 1

Os resultados indicaram que 40% dos especialistas escolheram a alternativa "Eu acho que as várias partes do guia estão muito bem integradas", enquanto 60% escolheram "Eu achei o guia fácil de usar".

As respostas mostram evidências iniciais de que o guia conseguiu integrar bem os passos necessários para configurar e operar o *pipeline*, mas pode se beneficiar de exemplos práticos e de uma organização ainda mais intuitiva para maximizar sua usabilidade.

4.4.2.2 A ordem em que os passos do Guia foram colocados é adequada?

As discussões revelaram um consenso entre os especialistas de que a ordem apresentada no Guia de DevOps é adequada, com 100% das opiniões indicando aprovação (Figura 29). Esse resultado mostra evidências iniciais de que o fluxo de passos foi bem estruturado, seguindo uma sequência lógica que facilita a implementação e configuração.

A ordem em que os passos do Guia foram colocados é adequada?

5 responses

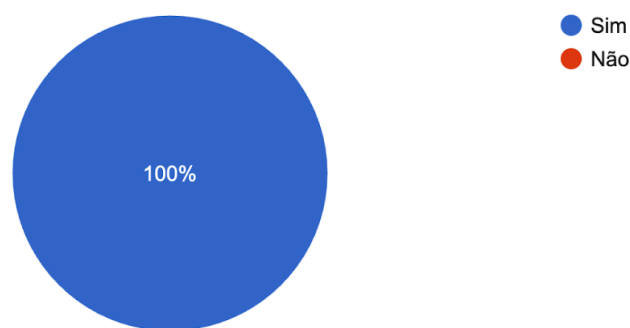


Figura 29: Resultados da pergunta 2

4.4.2.3 Principais pontos positivos do *pipeline* CI/CD e da containerização?

Durante as discussões, os especialistas destacaram diversos aspectos positivos da implementação do pipeline CI/CD e da containerização:

- **Automação de Processos:** Redução de tarefas manuais, agilizando o *deploy* e aumentando a eficiência.
- **Isolamento e Portabilidade:** Garantia de que os ambientes são consistentes e facilmente replicáveis.
- **Escalabilidade:** A infraestrutura containerizada facilita o crescimento do sistema sem grandes ajustes.
- **Controle sobre o Ambiente:** *Containers* permitem maior previsibilidade no funcionamento do sistema.

Comentários adicionais incluíram: "*Rodar a aplicação em containers faz com que tenhamos mais controle sobre o ambiente no qual a aplicação irá rodar. O uso de um registry próprio também é muito útil para rollback em casos de erros catastróficos.*"

Outro especialista destacou: *"A frequência de pushes de código com feedback imediato é uma das partes mais importantes de pipelines CI/CD. A integração contínua aumenta a confiança no deploy, e a containerização reduz as chances de falha, garantindo que, se funciona no ambiente de desenvolvimento, também funcionará em produção."*

4.4.2.4 Principais pontos negativos do *pipeline* CI/CD e da containerização?

As discussões também identificaram desafios e limitações na implementação atual:

- **Complexidade Inicial:** A configuração do *pipeline* e dos *containers* pode ser complicada, especialmente para novos desenvolvedores.
- **Dependência do Banco de dados Durante a *Build*:** Considerada uma limitação significativa, já que adiciona fragilidade ao processo de *build*.
- **Consumo de Recursos:** *Containers* que executam múltiplos serviços, como Maven e Tomcat, podem consumir muitos recursos, especialmente em máquinas de menor capacidade.

Outros comentários relevantes incluíram:

"Manter as três branches em um único arquivo de pipeline pode gerar confusão, embora centralize o processo."

"A única desvantagem que vejo é o tempo necessário para configurar o pipeline e educar desenvolvedores sobre o uso do Docker."

4.4.2.5 Sugestões para melhorar o *pipeline* ou a containerização?

Os especialistas propuseram melhorias importantes para o pipeline CI/CD e a containerização:

1. Otimização de Arquivos:

- Criar um `.dockerignore` para reduzir o tamanho das builds.
- Adicionar um `.gitignore` robusto, protegendo arquivos sensíveis e utilizando um `.env.example` como referência.

2. Revisão de Variáveis e Configurações:

- Remover variáveis redundantes, como `JAVA_OPTS` e `SELF_SSL`, onde o perfil `--prod` já define o ambiente.

3. Melhor Utilização do NFS:

- Abstrair caminhos de arquivos, evitando mapeamentos absolutos que podem quebrar em diferentes máquinas.

Um especialista também comentou:

"A dependência do banco de dados na build deveria ser eliminada, talvez com um banco mock ou ajustes no JOOQ. Isso tornaria o pipeline mais robusto e menos suscetível a falhas de conexão."

4.4.3 Considerações finais dos resultados

Os resultados das discussões com os especialistas destacaram os pontos fortes da implementação, como a eficiência, portabilidade e escalabilidade proporcionadas pela containerização e pela automação do *pipeline* CI/CD. A integração com ferramentas como o *Portainer* foi elogiada por simplificar a gestão e o *deploy* do sistema.

Por outro lado, foram apontadas limitações como a dependência do banco de dados durante o *build* e a ausência de testes automatizados no *pipeline*. Essas questões representam oportunidades importantes para melhorias futuras, reforçando a necessidade de ajustes técnicos e otimizações. As sugestões apresentadas, incluindo revisões na configuração e maior abstração de volumes, indicam um caminho claro para a evolução contínua do sistema, consolidando as bases estabelecidas por este trabalho.

5 Conclusão

O presente trabalho teve como objetivo a implantação de um processo de Integração e Deployment Contínuos (CI/CD) no Sistema de TCCs do departamento de INE da UFSC. Ao longo do desenvolvimento foi possível superar desafios relacionados à adaptação do sistema às novas tecnologias e processos, demonstrando o impacto positivo que práticas de automação e padronização podem proporcionar em sistemas acadêmicos.

Com base nos objetivos específicos traçados, cada etapa foi realizada de forma alinhada ao propósito principal. A análise da literatura sobre CI/CD (OE1) possibilitou uma compreensão mais profunda das práticas e ferramentas necessárias, enquanto a adaptação do guia pré-existente (OE2) permitiu ajustá-lo às particularidades do Sistema de TCCs. A implementação do processo de CI/CD (OE3) foi realizada com sucesso, promovendo maior consistência e automação nos fluxos de trabalho. Por fim, a etapa de implantação e avaliação (OE4) garantiu que o processo desenvolvido atendesse às expectativas, reforçando a confiabilidade e eficiência do sistema.

O trabalho também levantou indícios iniciais dos benefícios tangíveis trazidos pela automação, como a possível redução de erros humanos, a padronização de ambientes e a agilidade nas atualizações do sistema. Além disso, o uso de tecnologias como *Docker* e GitLab CI/CD foi essencial para atender às necessidades específicas levantadas durante a análise inicial, como a criação de um ambiente de desenvolvimento mais simples e o estabelecimento de *pipelines* eficientes.

Em conclusão, o projeto atingiu os objetivos propostos, mesmo que o ambiente de produção ainda não tenha sido atualizado devido a uma questão de cronograma de segurança, os resultados obtidos foram relevantes. Este trabalho deixa um caminho aberto para avanços futuros, tanto na evolução técnica do sistema quanto na aplicação de novas tecnologias que possam surgir.

5.1 Trabalhos futuros

Embora o trabalho tenha trazido avanços significativos para o Sistema de TCCs, existem algumas áreas que podem ser exploradas em futuras melhorias. As principais recomendações incluem:

- **Automação de testes unitários e de integração:** Apesar de o pipeline de CI/CD já automatizar as etapas de *build* e *deploy*, a inclusão de uma suíte robusta de testes automatizados garantiria maior confiabilidade e ajudaria a identificar problemas de forma mais ágil.
- **Expansão da documentação de DevOps:** A documentação criada pode ser complementada com mais detalhes sobre monitoramento contínuo, práticas de resolução de incidentes e orientações para manutenção de longo prazo. Isso permitirá que novos desenvolvedores se familiarizem rapidamente com o projeto.
- **Atualização de dependências do projeto:** Algumas dependências utilizadas no sistema, como a versão do MySQL e bibliotecas relacionadas, encontram-se desatualizadas e apresentam desafios de compatibilidade com infraestruturas modernas. A atualização dessas dependências é um passo essencial para garantir a segurança, a performance e a sustentabilidade do sistema.
- **Estudo de escalabilidade com Kubernetes:** A adoção de uma solução de orquestração como Kubernetes poderia trazer benefícios adicionais, como a possibilidade de escalar o sistema de forma dinâmica em períodos de alta demanda, garantindo maior estabilidade e disponibilidade.

Essas iniciativas podem fortalecer ainda mais o sistema, tornando-o mais preparado para atender às demandas futuras de seus usuários e desenvolvedores.

Referências

ANDRADE. **Implantação de um Processo de Integração e Deployment Contínuos em um Laboratório de Pesquisa**. 2022. Trabalho de Conclusão do Curso de Sistemas de Informação da Universidade Federal de Santa Catarina. Disponível em:

<https://repositorio.ufsc.br/bitstream/handle/123456789/243423/Monografia%20-%20Lucas%20Andrade.pdf?sequence=1&isAllowed=y>

BAPTISTA. **Desenvolvimento de Aplicativo Web Progressivo para Sistema de TCCs do INE/UFSC**. 2019. Trabalho de Conclusão do Curso de Sistemas de Informação da Universidade Federal de Santa Catarina. Disponível em:

<https://repositorio.ufsc.br/bitstream/handle/123456789/202477/TCC%20Orlando%20-%20Artigo%20e%20Cod%20Fonte%20Anexos.pdf?sequence=1&isAllowed=y>

BEECHAM, Sarah; HALL, Tracy; BRITTON, Carol; COTTEE, Michaela; RAINER, Austen. **Using an expert panel to validate a requirements process improvement model**. Journal of Systems and Software, v. 76, n. 3, p. 251–275, 2005. ISSN 0164-1212.

BOTELHO e UGIONI. **TCC UFSC - Nova Aplicação WEB para suporte à Coordenação de Projetos**. 2015. Disponível em:

<https://codigos.ufsc.br/antonio.c.mariani/TCC-UFSC/uploads/5c3353135e603f599b1354f912647921/monografia.pdf>

Colegiado de Sistemas de Informação. **UFSC**. 2011. Disponível em:

https://sin.paginas.ufsc.br/files/2011/09/RI-TCC-SIN_v15marco.pdf

DOCKER INC. **Docker**. 2023. Disponível em: <<https://docs.docker.com>>. Acesso em: Dezembro 2023.

DOLFING, Henrico. **Case Study 4: The \$440 Million Software Error at Knight Capital**.

Disponível em:

<https://www.henricodolfing.com/2019/06/project-failure-case-study-knight-capital.html>

DUVALL, P. M. **Continuous integration: patterns and antipatterns**. DZone refcard nº 84, 2010.

FOWLER, M. **Continuous integration**. 2006. Disponível em:

<https://martinfowler.com/articles/continuousIntegration.html>.

GITLAB CI/CD. **Gitlab**. 2023. Disponível em: <<https://docs.gitlab.com/ee/ci>>. Acesso em: Dezembro 2023.

GITLAB INC. **Gitlab**. 2023. Disponível em: <<https://gitlab.com>>. Acesso em: Dezembro 2023.

GLOVER, A.; DUVALL, P. M.; MATYAS, S. **Continuous Integration: Improving Software Quality and Reducing Risk**. Addison-Wesley, 2007.

HUMBLE, J.; FARLEY, D. **Continuous delivery: reliable software releases through build, test, and deployment automation**. Pearson Education, 2010.

GONÇALVES. **Melhoria da usabilidade do sistema de TCC do INE/UFSC do ponto de vista do aluno**. 2016. Trabalho de Conclusão do Curso de Sistemas de Informação da Universidade Federal de Santa Catarina. Disponível em: http://www.gqs.ufsc.br/files/2020/02/TCC_Diego_Fretta.pdf

LEAL. **Um módulo de recomendação de projetos baseado em machine learning para o sistema de gestão de TCC do INE**. 2022. Trabalho de Conclusão do Curso de Sistemas de Informação da Universidade Federal de Santa Catarina. Disponível em: https://repositorio.ufsc.br/bitstream/handle/123456789/243571/TCC_Ismael_modulo_recomendacao_a.pdf?sequence=1&isAllowed=y

MERKEL, Dirk. **Docker: lightweight linux containers for consistent development and deployment**. Linux Journal, v. 239, n. 2, p. 2, 2014.

MATTHIAS, Karl; KANE, Sean P. **Docker: Shipping Reliable Containers in Production**. Sebastopol: O'Reilly Media, Incorporated, 2015.

ONGHERO. **Implementação de Melhorias no Sistema de TCCs do Departamento de Informática e Estatística da UFSC: Perfil Professor**. 2018. Trabalho de Conclusão do Curso de Sistemas de Informação da Universidade Federal de Santa Catarina. Disponível em: https://repositorio.ufsc.br/bitstream/handle/123456789/192165/monografia_2018_final.pdf?sequence=1&isAllowed=y

PAULE, C. **Securing DevOps: detection of vulnerabilities in CD pipelines**. 2018. Dissertação (Mestrado) - Stuttgart University.

SKA, Yasmine; JANANI, P. **A study and analysis of continuous delivery, continuous integration in software development environment**. International Journal of Emerging Technologies and Innovative Research, v. 6, p. 96-107, 2019.

SYRJÄMÄKI, Joonas. **Exploring the advantages: A review of Docker container technology in the DevOps operating model**. 2023.

TURNBULL, James. **The Docker Book: Containerization is the new virtualization**. James Turnbull, 2014.

VAUCHER, S. **Comparing virtual machines and linux containers**. 2015. Disponível em: https://zenodo.org/record/47644/files/Comparing_Virtual_Machines_and_Linux_Containers_-_Final.pdf.

VASE, Tuomas. **Advantages of Docker**. 2015.

VASSALLO, Carmine; PALOMBA, Fabio; GALL, Harald C. **Continuous refactoring in CI: A preliminary study on the perceived advantages and barriers**. In: IEEE International Conference on Software Maintenance and Evolution (ICSME), 2018. IEEE, 2018.

ZHAO, Y.; SEREBRENIK, A.; ZHOU, Y.; FILKOV, V.; VASILESCU, B. **The impact of continuous integration on other software development practices: a large-scale empirical study**. In: Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering, ASE 2017, Urbana, IL, USA, 30 out. - 03 nov. 2017. Eds. G. Rosu, M. D. Penta, T. N. Nguyen. IEEE Computer Society, 2017. p. 60-71.

APÊNDICE A - ARTIGO: Modernização e Automação do Sistema de TCCs do INE/UFSC com CI/CD e Containerização

Pedro Schlickmann Mendes

Universidade Federal de Santa Catarina

mendes.pedro@grad.ufsc.br

Resumo. Este artigo apresenta o processo de modernização e automação do Sistema de Trabalhos de Conclusão de Curso (TCCs) do Departamento de Informática e Estatística (INE) da Universidade Federal de Santa Catarina (UFSC). A implementação foi realizada utilizando práticas de Integração Contínua (CI) e Deployment Contínuo (CD), integradas ao uso de containerização com Docker. O estudo aborda desde a análise dos problemas iniciais, como a ausência de automação nos deploys e a documentação insuficiente, até a proposta de uma solução que inclui pipelines automatizados no GitLab e um ambiente de homologação funcional gerenciado pelo Portainer. A documentação técnica desenvolvida também é destacada como um marco para futuras contribuições e manutenção do sistema. Os resultados mostram ganhos significativos em eficiência, escalabilidade e confiabilidade, além de apontar desafios técnicos e oportunidades para trabalhos futuros.

Abstract. This article presents the modernization and automation process of the Final Year Project (TCCs) System of the Department of Informatics and Statistics (INE) at the Federal University of Santa Catarina (UFSC). The implementation was carried out using Continuous Integration (CI) and Continuous Deployment (CD) practices, integrated with Docker containerization. The study covers the analysis of initial problems, such as the lack of deployment automation and insufficient documentation, and proposes a solution that includes automated pipelines in GitLab and a functional staging environment managed by Portainer. The developed technical documentation is also highlighted as a milestone for future contributions and system maintenance. The results demonstrate significant improvements in efficiency, scalability, and reliability, while identifying technical challenges and opportunities for future work.

1. Introdução

O Sistema de Trabalhos de Conclusão de Curso (TCCs) do Departamento de Informática e Estatística (INE) da Universidade Federal de Santa Catarina (UFSC) é uma ferramenta essencial para o gerenciamento de atividades acadêmicas relacionadas a projetos de TCC. Utilizado por alunos, orientadores e pela coordenação, o sistema desempenha funções críticas, como o registro de projetos, o acompanhamento de entregas e a gestão de avaliações. Contudo, com o passar dos anos, o aumento do uso e a evolução das práticas de desenvolvimento de software expuseram diversas limitações operacionais e estruturais no sistema.

Antes deste trabalho, o sistema era mantido em um ambiente com processos manuais de *deploy*, o que resultava em inconsistências e riscos de erros durante atualizações. Além disso, a ausência de automação no ciclo de vida do software dificultava a implementação de melhorias e a detecção de falhas antes da entrega em produção. Outro desafio significativo era a falta de uma documentação técnica abrangente, o que dificultava o *onboarding* de novos desenvolvedores e tornava o sistema menos acessível para contribuições externas ou melhorias estruturais.

A modernização do sistema foi conduzida com o objetivo de superar essas limitações, alinhando-se às melhores práticas de DevOps. O foco esteve na aplicação de Integração Contínua (CI) e Deployment Contínuo (CD), práticas amplamente reconhecidas por aumentar a eficiência, escalabilidade e confiabilidade de sistemas modernos. Para isso, foi implementado um pipeline automatizado no GitLab, que integrou etapas como *build*, *push* de imagens Docker para um registro centralizado e *deploy* automatizado em um ambiente de homologação gerenciado pelo *Portainer*. Paralelamente, foi realizada a containerização do sistema com *Docker*, garantindo maior consistência entre os ambientes de desenvolvimento, homologação e produção.

Outro aspecto central deste trabalho foi a criação de uma documentação técnica detalhada, consolidada em um Guia de DevOps. Este guia não apenas descreve as etapas de configuração e operação do sistema, mas também serve como referência para futuras contribuições, promovendo a sustentabilidade e a continuidade do projeto.

Este artigo explora os desafios técnicos enfrentados durante a implementação dessas soluções, detalha as melhorias alcançadas e analisa os impactos diretos no desempenho e na confiabilidade do sistema. Além disso, discute-se o potencial de evolução da infraestrutura atual, com propostas de trabalhos futuros voltados à inclusão de testes automatizados no pipeline, atualização de bibliotecas e eliminação de débitos técnicos. A modernização do Sistema de TCCs demonstrou que a aplicação de práticas modernas de DevOps pode transformar sistemas legados em soluções mais eficientes, escaláveis e alinhadas às demandas contemporâneas de desenvolvimento.

2. Proposta

A proposta deste trabalho consistiu em modernizar e automatizar os processos relacionados ao Sistema de Trabalhos de Conclusão de Curso (TCCs) do INE/UFSC, utilizando práticas de DevOps amplamente reconhecidas na literatura, como Integração Contínua (CI), Deployment Contínuo (CD) e containerização. A modernização foi planejada para resolver as principais limitações identificadas, tais como deploys manuais suscetíveis a erros, falta de automação no ciclo de desenvolvimento e deploy, e a inexistência de uma documentação técnica que consolidasse o conhecimento necessário para manutenção e evolução do sistema.

Para atender a esses objetivos, a proposta centrou-se em três pilares principais: a implementação de pipelines de CI/CD no GitLab, a containerização do sistema utilizando Docker e a criação de um ambiente de homologação funcional gerenciado pelo Portainer. Adicionalmente, foi elaborada uma documentação técnica detalhada, que serve tanto como um guia operacional quanto como uma base para futuras melhorias.

A primeira etapa da proposta envolveu a configuração de pipelines no GitLab CI/CD, divididos em estágios claros, como build, push e deploy. No estágio de build, imagens Docker foram geradas a partir do código-fonte e enviadas para um registro privado hospedado pela infraestrutura da UFSC. A estratégia de configuração incluiu a utilização de variáveis de ambiente para gerenciar credenciais de acesso e garantir a segurança dos dados. O estágio de deploy automatizado utilizou webhooks fornecidos pela SETIC, que permitiram a reinicialização dos serviços diretamente no Portainer, abstraindo a complexidade técnica do processo e aumentando a confiabilidade das atualizações.

A containerização do sistema foi um elemento central da proposta, garantindo consistência e portabilidade entre os ambientes. O uso de Docker permitiu encapsular todos os serviços do sistema, como o servidor de aplicação e o banco de dados, em contêineres isolados. Além disso, a configuração de volumes foi realizada para garantir a persistência de dados críticos, como arquivos estáticos e informações do banco de dados. Essa abordagem também facilita o rollback para versões anteriores, caso ocorra algum erro em produção.

Outro ponto relevante foi a criação de um ambiente de homologação funcional. Este ambiente foi configurado para validar alterações antes de sua promoção ao ambiente de produção, garantindo maior controle e segurança durante o ciclo de vida do software. A gestão dos containers no ambiente de homologação foi realizada por meio do Portainer, que forneceu uma interface intuitiva para monitoramento e controle das aplicações. O Portainer também facilitou a integração do pipeline de CI/CD, possibilitando a automatização de tarefas que anteriormente demandavam intervenção manual.

A proposta também incluiu a elaboração de um Guia de DevOps, consolidando as práticas e configurações realizadas. Este guia detalha os passos para configurar o ambiente local e em produção, explica a estrutura do pipeline de CI/CD e fornece orientações para resolver problemas comuns. O guia foi projetado para ser acessível a novos desenvolvedores, promovendo um onboarding mais rápido e eficiente.

Por fim, a proposta destacou a importância de adotar práticas que favorecem a escalabilidade e a sustentabilidade do sistema a longo prazo. Embora o foco inicial tenha sido a

automação e a modernização, a solução foi projetada para permitir futuras melhorias, como a inclusão de testes automatizados no pipeline e a eliminação de débitos técnicos, como a dependência do banco de dados durante o build. Com isso, a proposta se mostrou não apenas eficaz na resolução dos problemas existentes, mas também como uma base sólida para a evolução contínua do Sistema de TCCs.

3. Implementação

A modernização do Sistema de TCCs do INE/UFSC envolveu uma implementação detalhada e multifacetada, baseada em práticas modernas de DevOps. Este capítulo descreve cada uma das etapas técnicas e organizacionais, desde a configuração de pipelines de CI/CD no GitLab até a criação de um ambiente de homologação funcional, passando pela containerização com Docker e pela elaboração de uma documentação técnica robusta.

Configuração das Pipelines de CI/CD

A primeira etapa da implementação foi a criação de pipelines de CI/CD no GitLab, com o objetivo de automatizar e padronizar o ciclo de desenvolvimento e deploy do sistema. O arquivo `.gitlab-ci.yml` foi configurado para incluir três estágios principais:

- Build: Compilação do código e geração de imagens Docker.
- Push: Envio das imagens geradas para o registro Docker privado da UFSC.
- Deploy: Automação do processo de implantação, incluindo a reinicialização de serviços.

O estágio de build foi projetado para otimizar o tempo de execução e garantir a confiabilidade. Um dos pontos destacados foi a estratégia de copiar o arquivo `pom.xml` antes do restante dos arquivos, para que as dependências fossem baixadas antecipadamente, reduzindo o impacto de alterações no código durante o build. As imagens Docker geradas foram enviadas para um registro privado, utilizando variáveis de ambiente para configurar as credenciais de forma segura.

O estágio de deploy utilizou webhooks configurados no Portainer, que permitiram a reinicialização automatizada dos serviços. Essa abordagem eliminou a necessidade de intervenção manual no processo de atualização, reduzindo o tempo e os riscos associados a deploys manuais.

Containerização com Docker

A containerização foi um dos principais pilares da implementação, trazendo consistência e portabilidade ao sistema. O Docker foi utilizado para encapsular os serviços em containers isolados, abrangendo o servidor de aplicação, o banco de dados e outros componentes necessários. Isso garantiu que o sistema funcionasse de forma uniforme em diferentes ambientes, independentemente das configurações específicas de cada máquina.

Um aspecto crítico da containerização foi a configuração de volumes para gerenciar dados persistentes. O diretório `/home/tcc/files` foi mapeado para um compartilhamento NFS configurado pela SETIC, garantindo que os arquivos estáticos fossem armazenados de forma centralizada e permanecessem acessíveis mesmo após a reinicialização dos containers. Essa solução também facilitou o backup e a recuperação de dados, aumentando a resiliência do sistema.

Ambiente de Homologação

A criação de um ambiente de homologação funcional foi essencial para validar alterações antes de sua promoção ao ambiente de produção. Esse ambiente foi configurado para refletir o mais fielmente possível as condições reais de produção, utilizando o Portainer para gerenciar os containers e monitorar as operações.

Os containers foram configurados com *healthchecks*, que garantiram que o banco de dados estivesse operacional antes de iniciar o container da aplicação. Essa abordagem eliminou falhas de inicialização decorrentes de dependências não satisfeitas, aumentando a robustez do ambiente. Além disso, o uso do Portainer simplificou o gerenciamento dos serviços, oferecendo uma interface gráfica para realizar operações como reinicializações e atualizações.

Documentação Técnica

A elaboração de uma documentação técnica detalhada foi um componente essencial da implementação, garantindo que o conhecimento adquirido durante o trabalho fosse consolidado e acessível para futuras contribuições. Essa documentação foi organizada em um Guia de DevOps, que abrange desde a configuração dos ambientes local e de homologação até orientações práticas para resolver problemas comuns.

O guia incluiu instruções para configurar as variáveis de ambiente, detalhou a estrutura do pipeline de CI/CD no GitLab e forneceu explicações sobre o uso do Portainer para gerenciar os containers. Essa abordagem facilitou o onboarding de novos desenvolvedores e promoveu a continuidade do projeto, reduzindo a curva de aprendizado e centralizando o conhecimento técnico acumulado.

Além disso, o guia abordou práticas recomendadas, como o uso de volumes no Docker para gerenciar dados persistentes e estratégias para minimizar o tempo de build, tornando o sistema mais eficiente e fácil de manter.

Desafios Técnicos

Durante a implementação, alguns desafios técnicos significativos foram enfrentados. Um dos mais importantes foi a dependência do banco de dados durante o processo de build. Essa dependência, causada pela integração do JOOQ para a geração de código, exigiu ajustes no pipeline e um cuidado especial para garantir que o banco de dados estivesse acessível durante a compilação. Embora funcional, essa configuração representa um débito técnico que deve ser revisado em trabalhos futuros.

Outro desafio foi a complexidade inicial da configuração, que exigiu uma coordenação cuidadosa com a SETIC para o mapeamento de volumes NFS e a criação de webhooks no Portainer. Apesar dessas dificuldades, as soluções implementadas se mostraram robustas e alinhadas às melhores práticas de DevOps.

Resultados da Implementação

Os resultados da implementação demonstraram ganhos significativos em eficiência e confiabilidade. O tempo necessário para deploys foi reduzido de mais de uma hora para menos de

cinco minutos, eliminando falhas associadas a inconsistências entre os ambientes. A adoção de pipelines automatizados e a integração com o Portainer simplificaram o ciclo de vida do software, garantindo que as atualizações fossem realizadas de maneira segura e eficiente.

A containerização também trouxe benefícios claros, permitindo maior portabilidade e escalabilidade. O sistema agora pode ser replicado em diferentes ambientes sem a necessidade de ajustes manuais, o que é particularmente relevante para equipes distribuídas ou futuras migrações para ambientes em nuvem.

A criação do ambiente de homologação funcional e a elaboração do Guia de DevOps consolidaram as melhorias realizadas, estabelecendo uma base sólida para a evolução contínua do sistema. Essa infraestrutura permite validar alterações com maior rigor, reduzindo os riscos de falhas em produção e promovendo um ciclo de desenvolvimento mais confiável.

4. Conclusão

A modernização e automação do Sistema de TCCs do INE/UFSC representaram um avanço significativo na eficiência e confiabilidade do ciclo de desenvolvimento e deploy do sistema. Por meio da adoção de práticas de DevOps como Integração Contínua (CI), Deployment Contínuo (CD) e containerização com Docker, foi possível superar limitações estruturais e operacionais, além de estabelecer uma base sólida para futuras melhorias.

A implementação de pipelines automatizados no GitLab eliminou processos manuais suscetíveis a erros, reduzindo o tempo necessário para deploys de mais de uma hora para poucos minutos. A containerização garantiu consistência entre os ambientes de desenvolvimento, homologação e produção, permitindo maior portabilidade e escalabilidade. A configuração de healthchecks e volumes persistentes contribuiu para a robustez do sistema, enquanto o ambiente de homologação funcional facilitou a validação rigorosa das alterações antes de sua promoção à produção.

Outro marco importante foi a elaboração de uma documentação técnica abrangente consolidada em um Guia de DevOps. Esse guia não apenas detalhou as práticas e configurações realizadas, mas também promoveu a acessibilidade do sistema para novos desenvolvedores, reduzindo a curva de aprendizado e garantindo a continuidade do projeto.

Apesar das conquistas, alguns desafios técnicos ainda precisam ser abordados. A dependência do banco de dados durante o processo de build, por exemplo, é um débito técnico que requer revisão para tornar o sistema ainda mais independente e robusto. Além disso, a ausência de testes automatizados no pipeline é um ponto a ser considerado em trabalhos futuros, visando aumentar a confiabilidade em cenários mais complexos.

Este trabalho destacou como a aplicação de práticas modernas de DevOps pode transformar sistemas legados, resolvendo problemas imediatos e criando oportunidades para inovação contínua. A infraestrutura atual do Sistema de TCCs posiciona o INE/UFSC como um exemplo de adoção bem-sucedida de tecnologias modernas no contexto acadêmico. Mais do que uma modernização técnica, este trabalho reflete o compromisso com a melhoria de processos essenciais para a gestão acadêmica e o suporte a todos os envolvidos no ciclo de TCCs.

Com a base estabelecida, o sistema está preparado para evoluções futuras que podem incluir a implementação de testes automatizados, melhorias na segurança e simplificação de sua infraestrutura. Esse caminho demonstra o impacto duradouro da modernização, tanto no presente quanto nas possibilidades de crescimento e inovação nos próximos anos.