Documentação de Projeto

para o sistema

HookCI

Versão 1.0

Projeto de sistema elaborado pelo(s) aluno(s) Gabriel Dolabela Marques e Henrique Carvalho Almeida e apresentado ao curso de **Engenharia de Software** da **PUC Minas** como parte do Trabalho de Conclusão de Curso (TCC) sob orientação de conteúdo da professora Joana Gabriela Ribeiro de Souza, orientação acadêmica do professor Cleiton Silva Tavares e orientação de TCC II do professor (a ser definido no próximo semestre).

06/04/2025

Tabela de Conteúdo

Histórico de Revisões

Nome	Data	Razões para Mudança	Versão
Gabriel Dolabela Henrique Almeida	06/04	Criação do documento	0.1.0
Gabriel Dolabela Henrique Almeida	21/04	Adição de Diagramas de Sequência, Classes e Comunicação	0.2.0
Henrique Almeida	23/04	Correções da criação do documento	0.2.1
Gabriel Dolabela Henrique Almeida	06/05	Adição de Arquitetura Lógica, Diagramas de Estados, Diagramas de Componentes, Diagramas de Implantação, do modelo de configuração e do glossário	0.3.0

1. Introdução

Este documento agrega: 1) a elaboração e revisão de modelos de domínio e 2) modelos de projeto para o sistema HookCI. A referência principal para a descrição geral do problema, domínio e requisitos do sistema é este documento de especificação que descreve a visão de domínio do sistema. Tal especificação acompanha este documento. Anexo a este documento também se encontra o Glossário.
HookCI consiste em uma ferramenta para a execução da Integração Contínua (CI, do inglês *Continuous Integration*) de forma local, integrando Docker e Git *hooks* para validar *commits* e *pushes*.
A proposta se diferencia dos serviços de CI tradicionais, como GitHub Actions ou GitLab CI, pois traz o conceito de CI para o ambiente local, reduzindo o tempo de retorno e evitando a sobrecarga em servidores remotos. Assim, a ferramenta atua como uma camada preventiva que assegura que somente códigos que passam nos testes sejam integrados ao repositório central.
Outras ferramentas como Husky executam os comandos diretamente no ambiente local. Pretende-se isolar os testes em Docker para garantir a consistência, oferecendo configuração via YAML e uma Interface de Linha de Comando (CLI, do inglês *Command Line Interface*) dedicada, tornando o processo de CI local mais estruturado.

2. Modelos de Usuário e Requisitos

2.1 Descrição de Atores

Desenvolvedor: É o ator principal, responsável por implementar funcionalidades, escrever código e executar testes. Utiliza o HookCI para validar *commits* e *pushes*, garantindo que somente alterações que passam na suíte de testes sejam integradas ao repositório. O ator se beneficia do retorno imediato proporcionado pela execução dos testes em ambiente local, que utiliza Docker para isolamento e consistência.

Desenvolvedor / Administrações Técnico: Atua na configuração e na manutenção do ambiente de desenvolvimento e dos parâmetros de execução da ferramenta. É responsável por definir, por meio de arquivos YAML, as regras e os comandos de teste a serem utilizados pelo HookCI.

2.2 Modelos de Usuários

Para melhor compreender as necessidades e expectativas dos usuários do HookCI, foram criadas personas que representam os perfis de uso da ferramenta. A seguir, são apresentadas duas personas que englobam os principais perfis de desenvolvedores e administradores envolvidos.
A Tabela 1 descreve a persona do usuário Márcio Nunes, um desenvolvedor iniciante que deseja ter um retorno rápido e assertivo relativo à qualidade de seu código.

	Márcio Nunes, Desenvolvedor Júnior
Descrição	Márcio tem 23 anos e está iniciando sua carreira como desenvolvedor. Trabalha em uma pequena equipe de desenvolvimento e precisa de ferramentas que facilitem o aprendizado e ofereçam retorno rápido sobre a qualidade do código.
Objetivos	• Obter respostas imediatas sobre possíveis falhas em seus *commits*, permitindo aprendizado contínuo. • Adquirir confiança na integração de novas funcionalidades sem comprometer a integridade do repositório.
Dores	• Dificuldade em identificar rapidamente os erros decorrentes de testes mal sucedidos. • Processos de CI que, por serem remotos, atrasam seu fluxo de trabalho.

Tabela 1. Persona Márcio

O sistema HookCI fornece retorno imediato através da execução dos testes localmente e isolando o ambiente de testes com Docker, garantindo consistência e evitando conflitos com as configurações locais.
A Tabela 2 descreve a persona do usuário Paulo Lopes, um desenvolvedor administrador de um time que deseja garantir a qualidade de seu projeto a todo momento de forma automatizada e padronizada.

	Paulo Lopes, Desenvolvedor Sênior
Descrição	Paulo tem 29 anos e atua como desenvolvedor sênior em uma equipe de médio porte. Além de codificar, ele também assume funções de administração técnica, configurando o ambiente de desenvolvimento e definindo as diretrizes para a integração contínua.
Objetivos	• Estabelecer um fluxo de trabalho que minimize falhas e garanta a qualidade do código entregue. • Otimizar o tempo de execução dos testes, mantendo um ciclo de retorno rápido e eficiente.

Dores	• Dificuldade em padronizar os ambientes de desenvolvimento entre os membros da equipe. • Gerenciar configurações de testes que, quando executados em ambientes distintos, podem apresentar resultados inconsistentes.

Tabela 2. Persona Paulo

O sistema HookCI possibilita configuração centralizada via arquivos YAML, facilitando a instalação automatizada dos Git *hooks* e isolando os testes em containers Docker, assegurando que todos os membros da equipe trabalhem com o mesmo ambiente de CI, independentemente das particularidades de suas máquinas.

2.3 Modelo de Casos de Uso e Histórias de Usuários

Esta subseção apresenta o diagrama de casos de uso e as histórias de usuários que guiam o desenvolvimento da aplicação HookCI.

2.3.1 Diagrama de Casos de Uso

A Figura 1 ilustra o diagrama de casos de uso do sistema HookCI, no qual são representados três atores: Desenvolvedor, Administrador e o sistema Git. O Desenvolvedor interage com a documentação e inicia a execução de testes por meio da CLI; o Administrador realiza a inicialização do projeto na ferramenta; e o sistema Git, por sua vez, dispara a execução dos testes automaticamente via Git *hooks*.

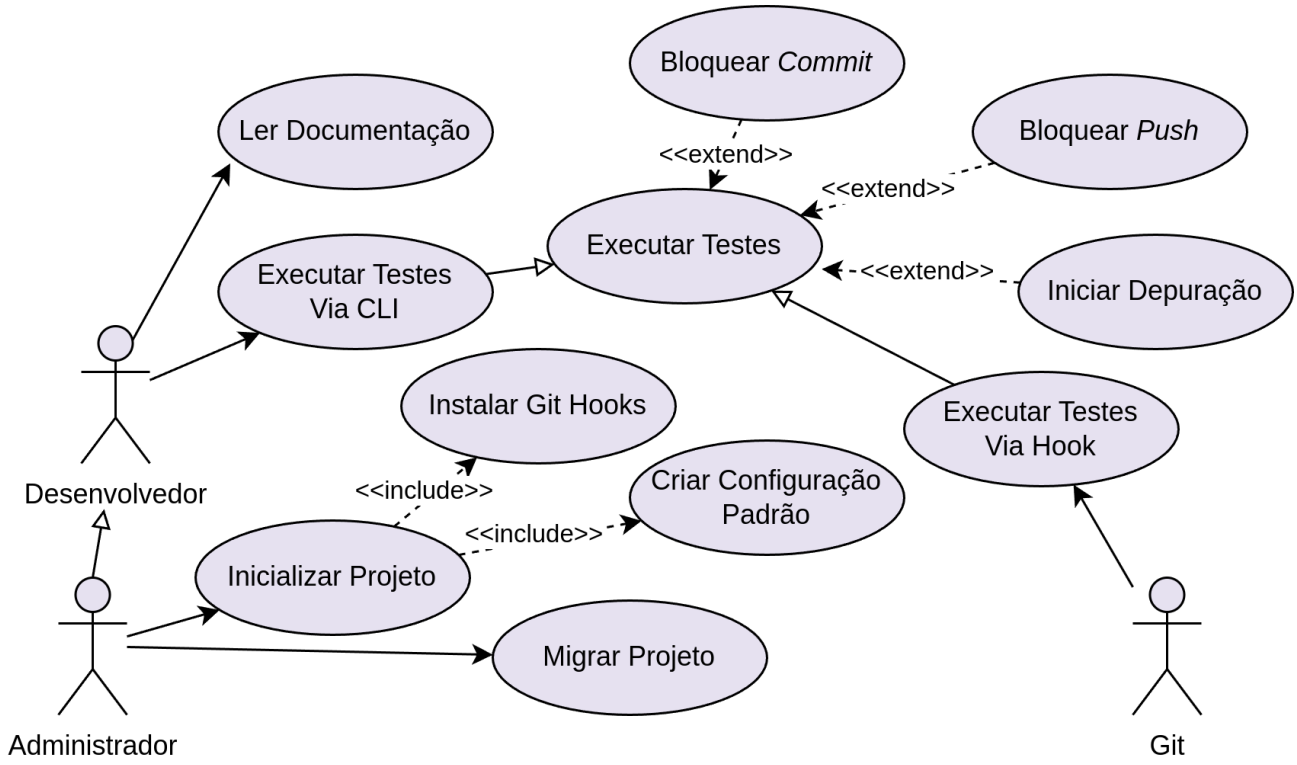

Figura 1. Diagrama de caso de uso

2.3.2 História de Usuários

US01:

Como desenvolvedor, desejo acessar a documentação dos comandos disponíveis via CLI, para entender como utilizar a ferramenta corretamente

US02:

Como administrador, desejo inicializar um projeto com o HookCI, para configurar o ambiente local e preparar a estrutura básica de integração contínua.

US03:

Como administrador, desejo que seja criado um arquivo YAML de configuração de forma automática, para definir os parâmetros de execução de testes no projeto.

US04:

Como administrador, desejo que os *hooks* do Git sejam instalados automaticamente no projeto, para evitar configurações manuais propensas a erro.

US05:

Como administrador, desejo editar o arquivo de configuração YAML, para personalizar comandos, variáveis de ambiente e políticas de execução.

US06:

Como administrador, desejo migrar a configuração pré-existente para a mais atual da ferramenta de forma automática.

US07:

Como administrador, desejo ser alertado pela ferramenta caso a configuração não possa ser automaticamente migrada.

US08:

Como desenvolvedor, desejo que os testes sejam executados automaticamente ao realizar um *commit* ou *push*, para evitar que código defeituoso seja integrado ao repositório.

US09:

Como desenvolvedor, desejo executar testes manualmente via CLI, para verificar o comportamento do sistema sob demanda.

US10:

Como desenvolvedor, desejo que os testes sejam executados em um ambiente Docker isolado, para garantir a consistência entre ambientes de desenvolvimento.

US11:

Como desenvolvedor, desejo visualizar os logs da execução de testes na linha de comando em caso de erro, a fim de entender o que foi executado e identificar falhas rapidamente.

US12:

Como desenvolvedor, desejo ajustar o nível de verbosidade dos logs da CLI, para escolher entre mensagens mais detalhadas ou mais concisas conforme a situação.

US13:

Como desenvolvedor, desejo ter acesso a um modo de depuração em caso de falha nos testes, para investigar interativamente o ambiente de execução e entender a origem dos problemas.

US14:

Como desenvolvedor, desejo validar a estrutura e os valores do arquivo YAML de configuração, para garantir que ele esteja correto antes da execução.

US15:

Como desenvolvedor, desejo configurar testes não essenciais que apenas avisem o desenvolvedor caso falhem.

US16:

Como desenvolvedor, desejo configurar uma imagem Docker personalizada através de um nome e versão.

US17:

Como desenvolvedor, desejo utilizar um *Dockerfile* para definir o ambiente de testes, para maior controle sobre dependências e ferramentas instaladas.

US18:

Como desenvolvedor, desejo montar automaticamente o diretório atual do repositório no contêiner durante a execução dos testes, para garantir que o código-fonte correto seja testado.

2.4 Diagrama de Sequência do Sistema e Contrato de Operações

Nesta subseção é apresentado o Diagrama de Sequência do Sistema (DSS) e os Contratos de Operações.

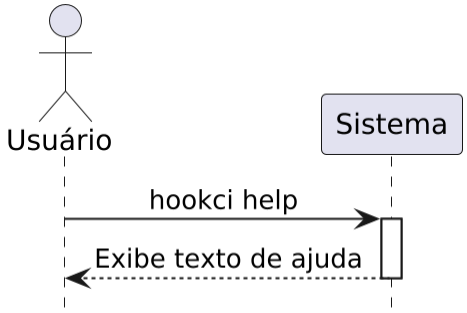

Figura 2. DSS para a operação “Mostrar ajuda”

Contrato	Mostrar ajuda

Operação	display_help()
Referências cruzadas	História de usuário: US01.
Pré-condições	Ferramenta instalada no ambiente.
Pós-condições	

Tabela 3. Contrato da operação “Mostrar ajuda”

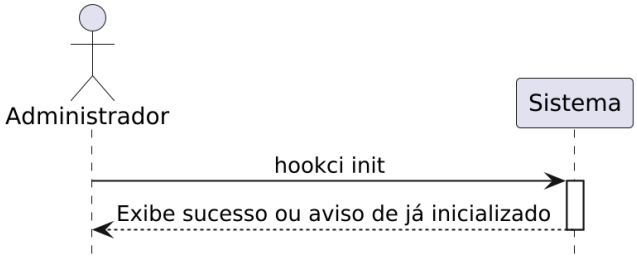

Figura 3. DSS para a operação “Inicializar projeto”

Contrato	Inicializar projeto
Operação	init_project()
Referências cruzadas	Histórias de usuário: US02, US03, US04.
Pré-condições	Ferramenta instalada no ambiente, Projeto Git na pasta atual ou mãe, Docker instalado.
Pós-condições	Ferramenta instalada no projeto.

Tabela 4. Contrato da operação “Inicializar projeto”

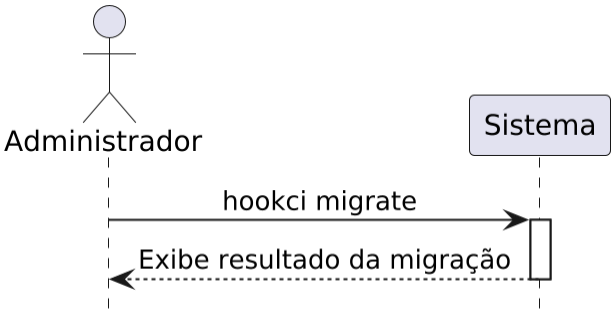

Figura 4. DSS para a operação “Migrar configuração”

Contrato	Migrar configuração
Operação	migrate_config()
Referências cruzadas	Histórias de usuário: US06, US07.
Pré-condições	Ferramenta instalada no ambiente e no projeto, Projeto Git na pasta atual ou mãe.
Pós-condições	Configuração atualizada para sua última versão.

Tabela 5. Contrato da operação “Migrar configuração”

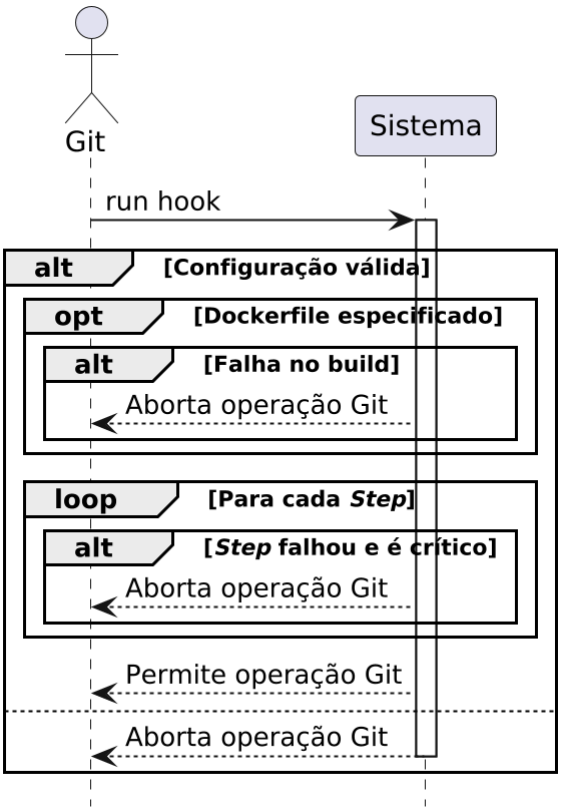

Figura 5. DSS para a operação "Executar CI via *Hook*"

Contrato	Executar CI via *Hook*
Operação	run_hook()
Referências cruzadas	Histórias de usuário: US05, US08, US10, US11, US14, US15, US16, US17, US18.
Pré-condições	Ferramenta instalada no ambiente e no projeto, Projeto Git na pasta atual ou mãe, configuração de projeto desatualizada.
Pós-condições	Configuração atualizada.

Tabela 6. Contrato da operação "Executar CI via *Hook*"

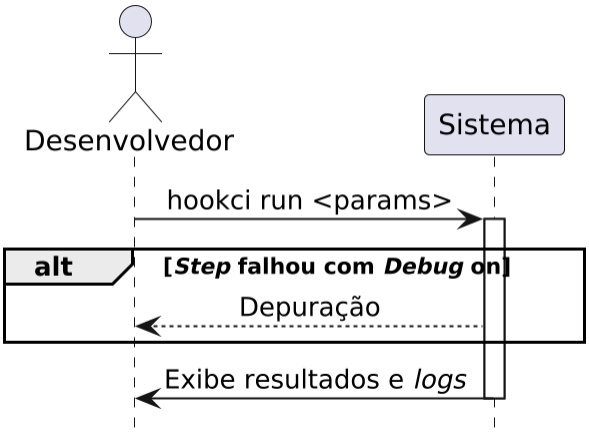

Figura 6. DSS para a operação “Executar CI”

Contrato	Executar CI
Operação	run()
Referências cruzadas	Histórias de usuário: US05, US09, US10, US11, US12, US13, US14, US15, US16, US17, US18.
Pré-condições	Ferramenta instalada no ambiente e no projeto, Projeto Git na pasta atual ou mãe, configuração de projeto desatualizada.
Pós-condições	Configuração atualizada.

Tabela 7. Contrato da operação “Executar CI”

3. Modelos de Projeto

3.1 Diagrama de Classes

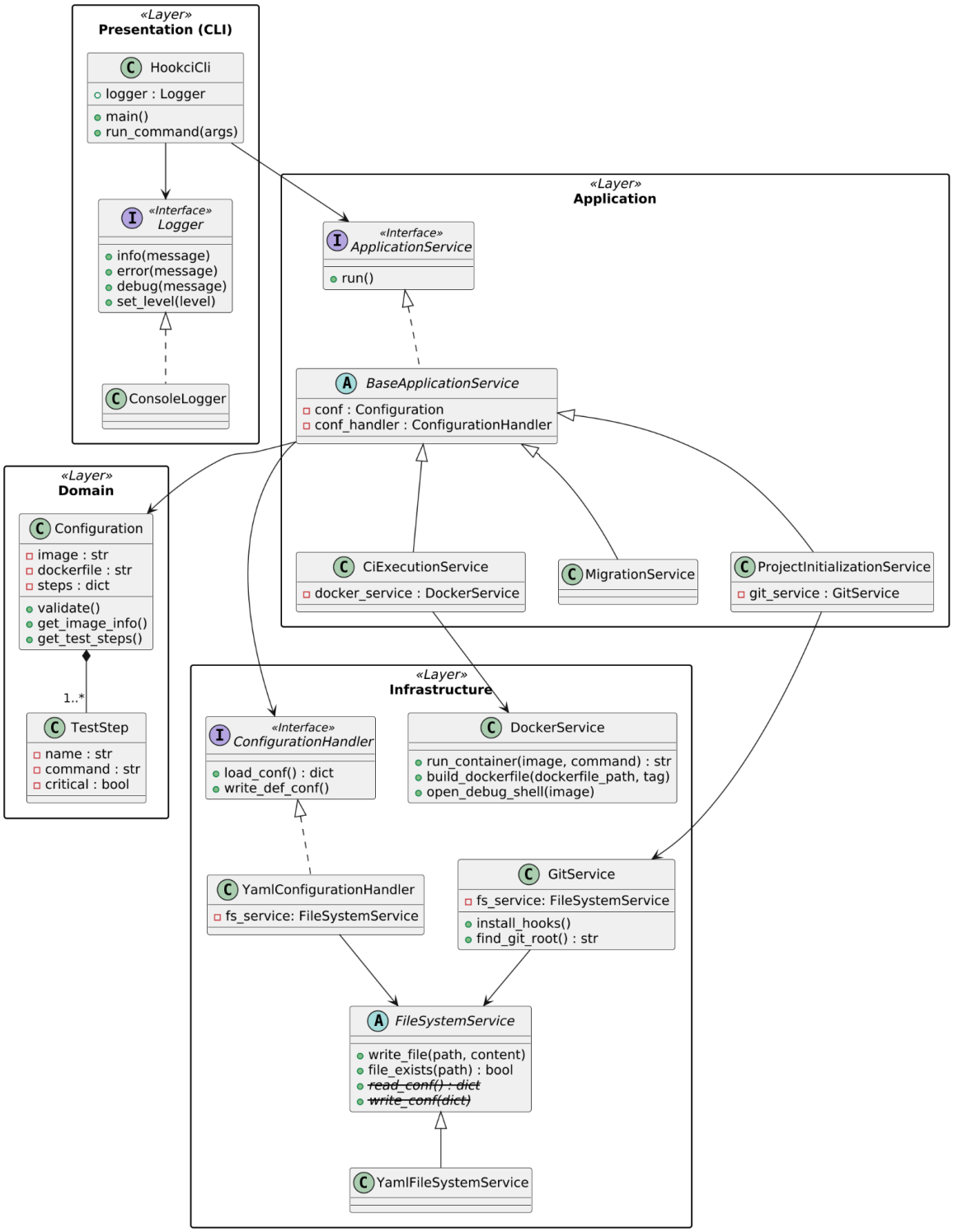

Figura 7. Diagrama de Classes do sistema

A Figura 7 detalha a estrutura estática do sistema HookCI. As classes são organizadas em camadas arquiteturais: *Presentation*, responsável pela CLI; *Application*, que orquestra os serviços e as histórias de usuário; *Domain*, contendo a lógica de negócio central e entidades como Configuration e TestStep; e *Infrastructure*, que gerencia interações externas com o sistema Git, a plataforma Docker e o sistema de arquivos. O diagrama ilustra as classes principais, suas interfaces e os relacionamentos entre elas, como herança, composição e dependência. Este modelo serve como um mapa dos componentes de software e de como eles se conectam para formar o sistema.

3.2 Diagramas de Sequência

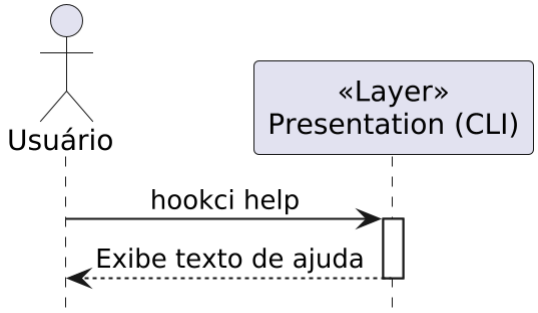

Figura 8. Diagrama de Sequência para a operação “Mostrar ajuda”

A Figura 8 descreve a sequência para exibir a ajuda ao usuário. O Usuário executa o comando “hookci help” na CLI. A camada de *Presentation* processa esta solicitação e responde diretamente ao Usuário, exibindo o texto de ajuda formatado.

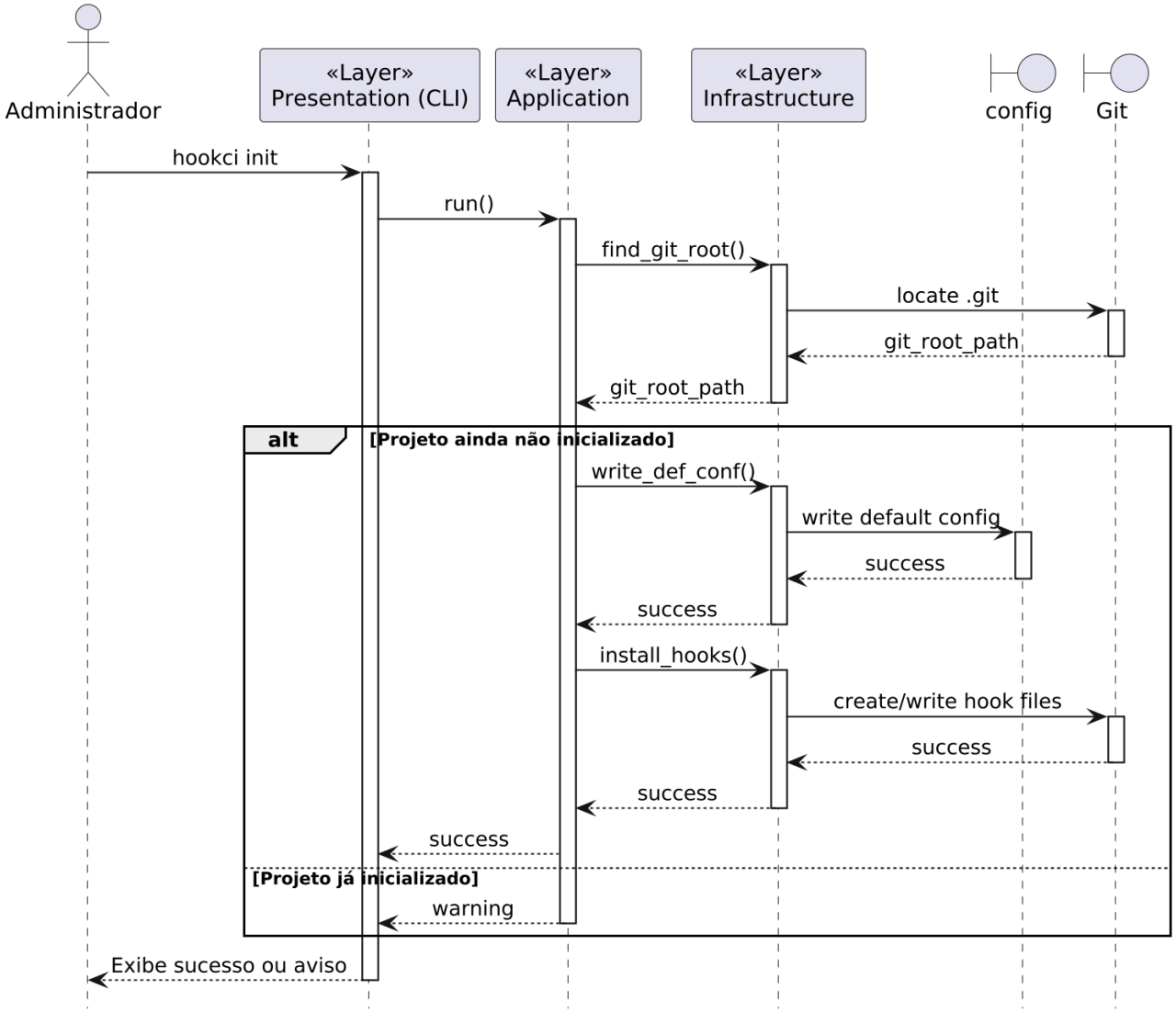

Figura 9. Diagrama de Sequência para a operação “Inicializar projeto”

A Figura 9 ilustra o fluxo de inicialização do projeto, acionado pelo comando “hookci init” executado pelo Administrador. A CLI encaminha a requisição para a camada de *Application*. Esta camada coordena com a camada de *Infrastructure* para realizar duas ações principais: primeiro, localizar a raiz do repositório Git e, caso o projeto ainda não tenha sido inicializado, criar o arquivo de configuração padrão; segundo, instalar os *scripts* de *hook* necessários no diretório .git/ do repositório. O resultado da operação, sucesso ou um aviso caso o projeto já esteja inicializado, é então comunicado ao Administrador pela CLI.

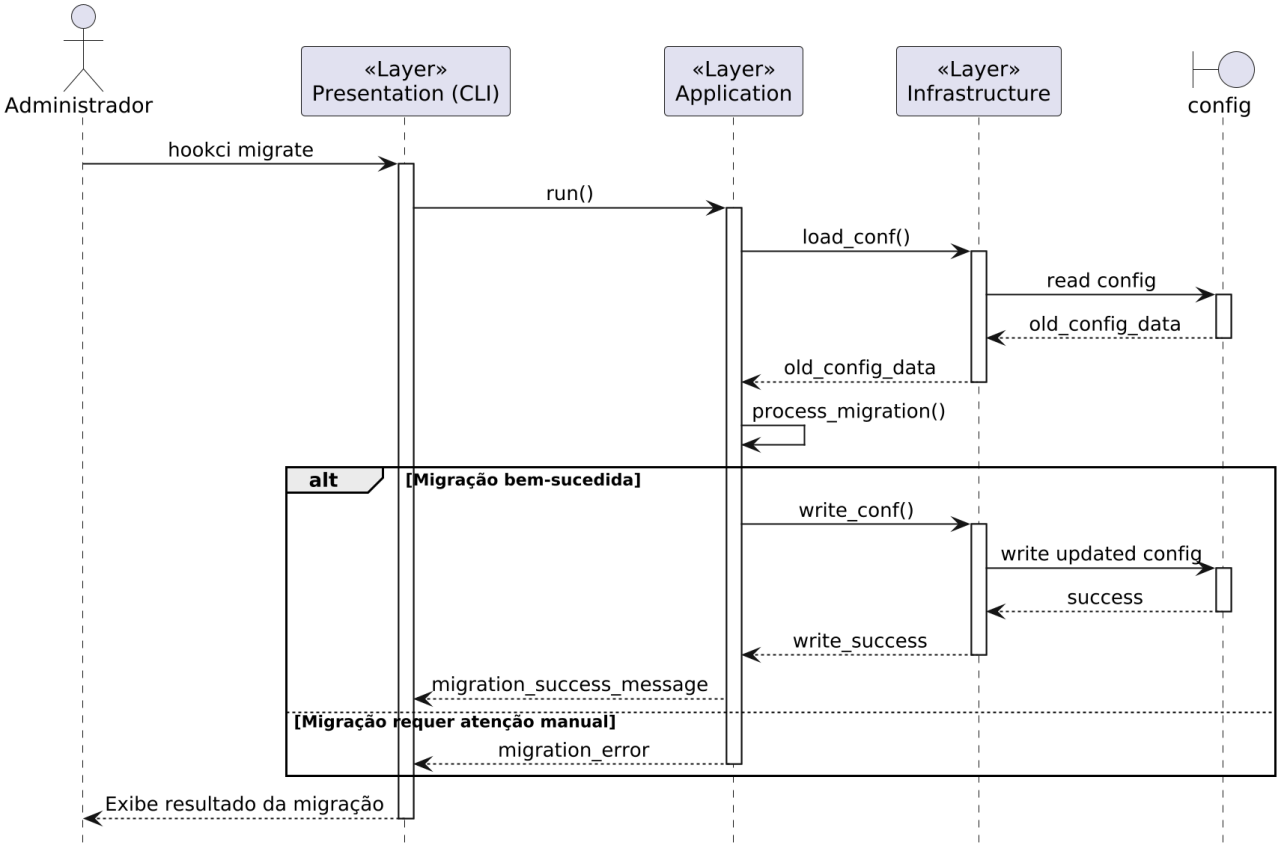

Figura 10. Diagrama de Sequência para a operação “Migrar configuração”

A Figura 10 detalha o processo de migração da configuração, iniciado pelo Administrador com o comando “hookci migrate”. A camada de *Application* solicita à camada de *Infrastructure* a leitura do arquivo de configuração existente. Em seguida, a própria camada de *Application* processa os dados lidos para convertê-los ao formato mais recente. Se a migração for possível e bem-sucedida, a *Application* instrui a *Infrastructure* a escrever o novo arquivo de configuração. Por fim, a CLI informa o Administrador sobre o sucesso ou eventuais problemas na migração.

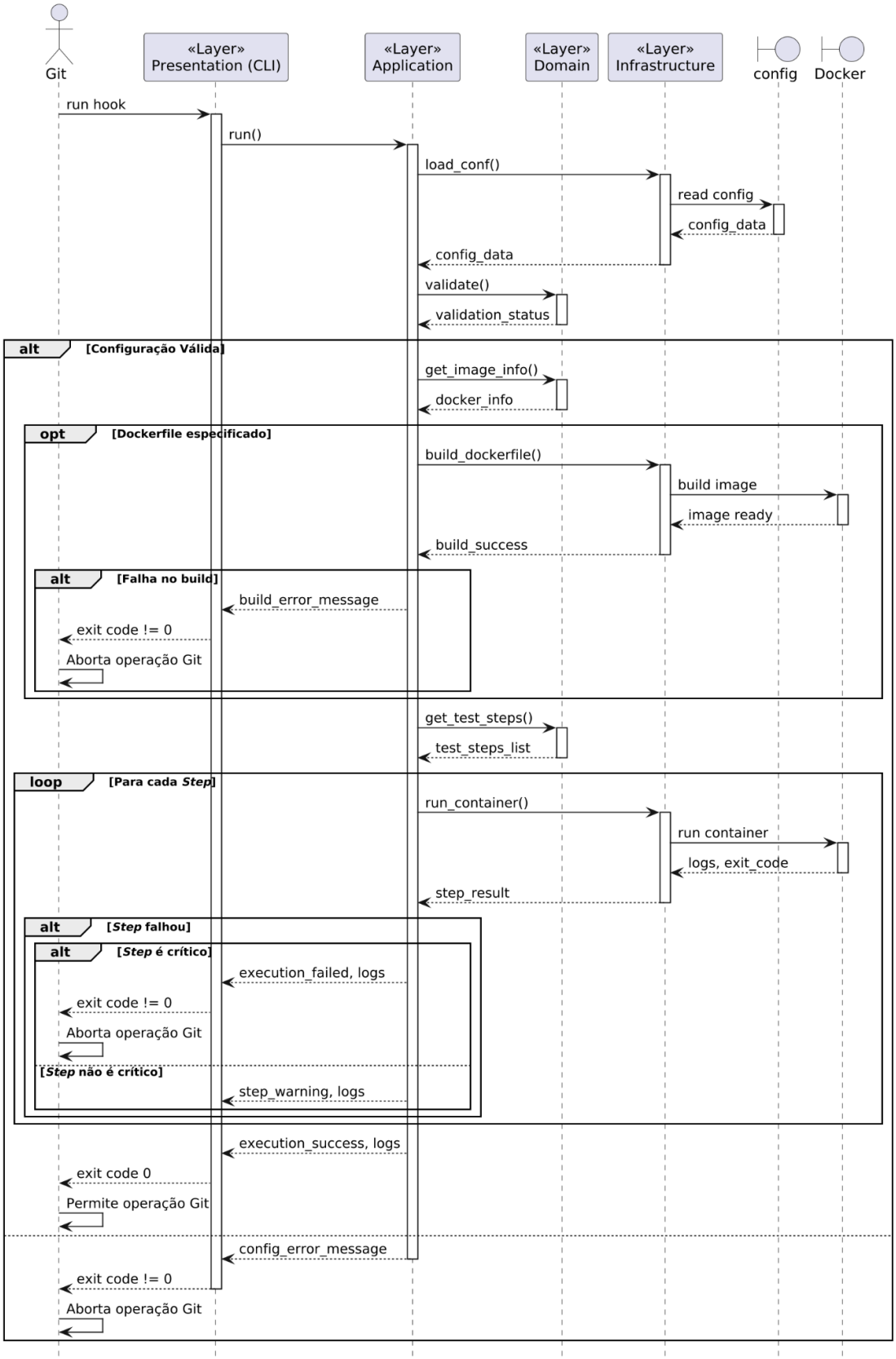

Figura 11. Diagrama de Sequência para a operação “Executar CI via *Hook*”

A Figura 11 demonstra a execução automática da CI, disparada por um *hook* do sistema Git, como o *pre-commit*. O Git executa o *script* do *hook*, que por sua vez invoca a CLI do HookCI. A camada de *Application* é acionada e, interagindo com as camadas de *Domain* e *Infrastructure*, carrega e valida a configuração do config.yaml; obtém as informações sobre o ambiente Docker, seja uma imagem pré-existente ou um *Dockerfile* a ser construído; executa sequencialmente cada etapa de teste definida na configuração dentro de contêineres Docker isolados. Ao final, a CLI retorna um código de saída para o Git: 0 indica sucesso, permitindo que a operação Git continue; um código diferente de 0 indica falha em uma etapa crítica, fazendo com que a operação Git seja abortada.

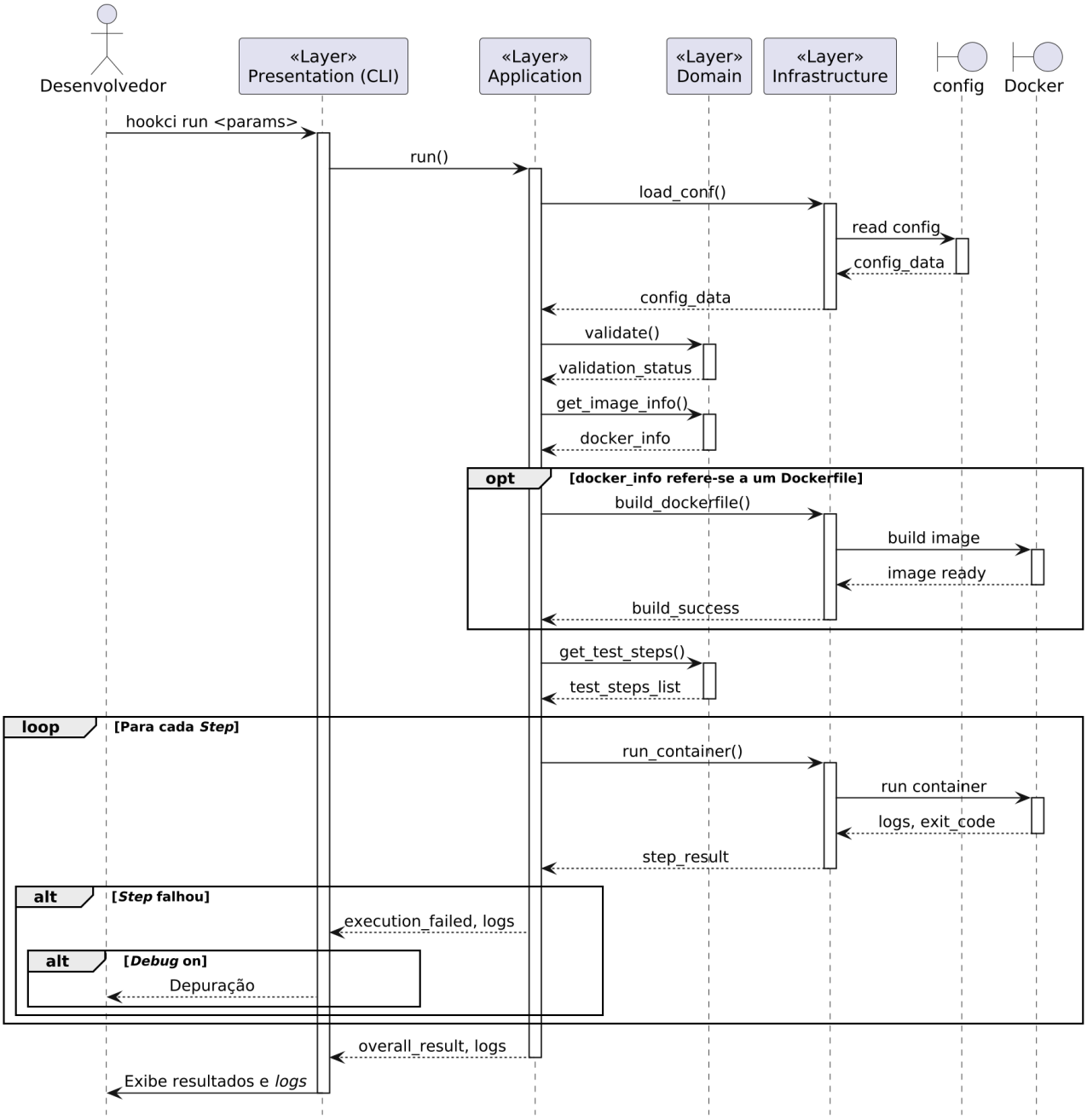

Figura 12. Diagrama de Sequência para a operação “Executar CI”

A Figura 12 representa a execução manual da CI, iniciada pelo Desenvolvedor através do comando "hookci run" na CLI. O fluxo de execução das etapas de teste é semelhante ao disparado pelo *hook* Git, envolvendo as camadas de *Application*, *Domain* e *Infrastructure* para validar a configuração, gerenciar o ambiente Docker e executar os testes em contêineres. A principal diferença reside no início e fim da interação: ela é explicitamente solicitada pelo Desenvolvedor e o resultado final, incluindo *logs* e o status de sucesso ou falha, é exibido diretamente no terminal para o Desenvolvedor, sem interferir no fluxo de uma operação Git.

3.3 Diagramas de Comunicação

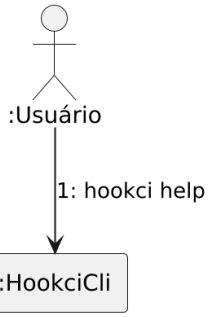

Figura 13. Diagrama de Comunicação para a operação "Mostrar ajuda"

A Figura 13 mostra a interação para exibir a ajuda. O ator Usuário interage com a instância HookciCli da camada de apresentação, que responde diretamente.

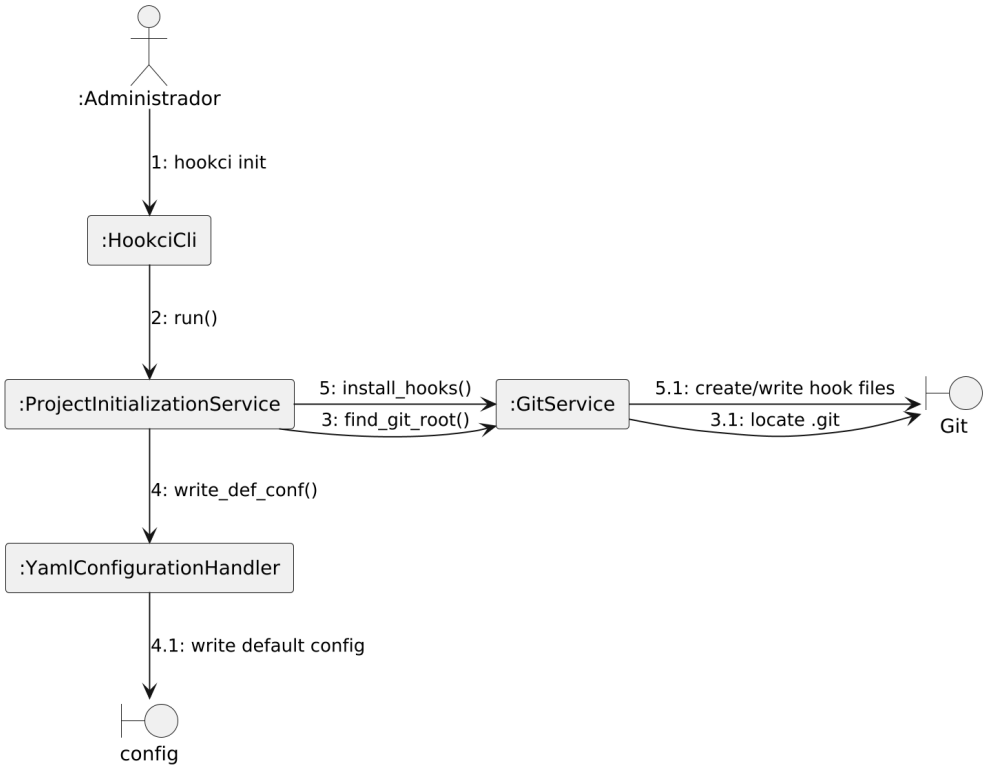

Figura 14. Diagrama de Comunicação para a operação "Inicializar projeto"

A Figura 14 detalha as colaborações para inicializar um projeto. O Administrador aciona HookciCli, que delega para ProjectInitializationService. Este serviço interage com GitService e com YamlConfigurationHandler para configurar o projeto e instalar os *hooks*.

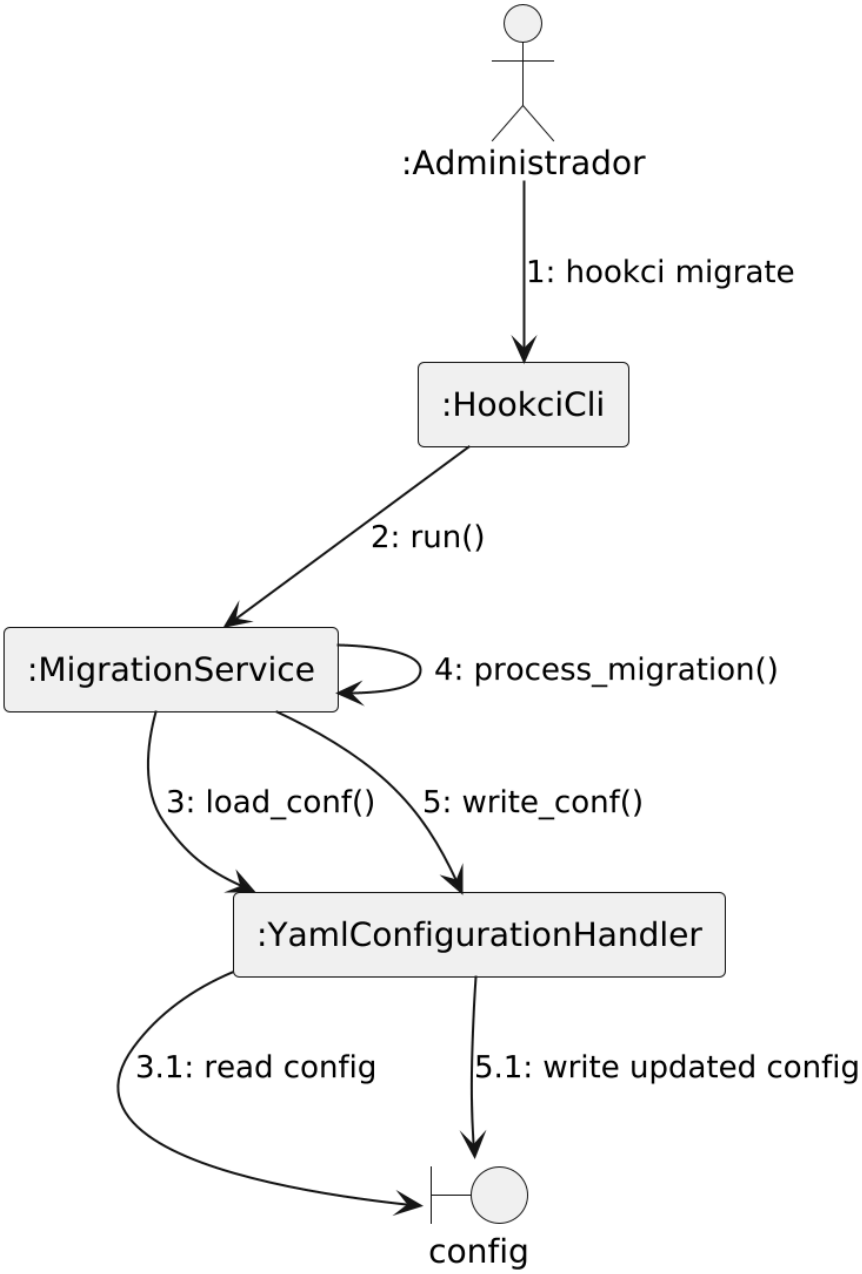

Figura 15. Diagrama de Comunicação para a operação "Migrar configuração"

A Figura 15 ilustra as interações na migração de configuração. O Administrador inicia o processo via HookciCli, que chama o MigrationService. Este serviço usa o YamlConfigurationHandler para ler e escrever no ConfigFile, após processar a migração internamente.

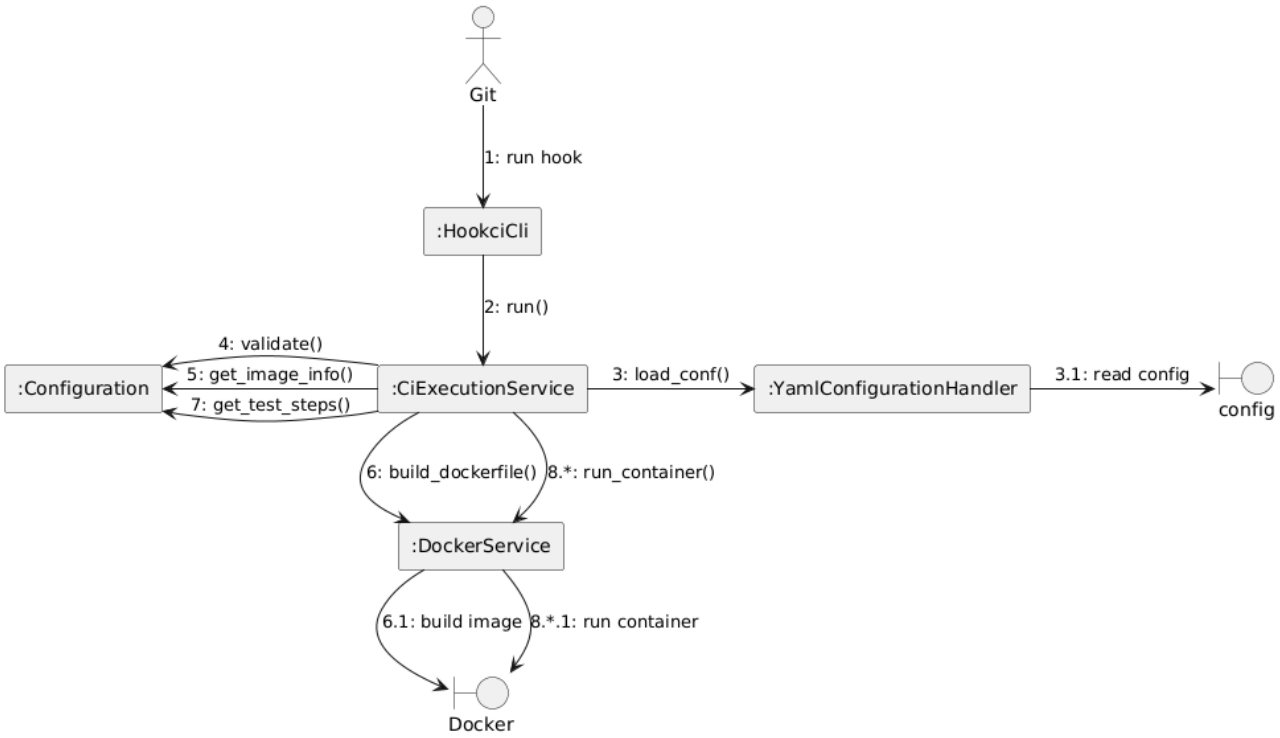

Figura 16. Diagrama de Comunicação para a operação "Executar CI via *hook*"

A Figura 16 mostra a colaboração quando a CI é disparada por um *hook* do Git. O *hook* executa HookciCli, que invoca CiExecutionService. O serviço coordena a leitura da configuração, validação e obtenção de informações, construção opcional de imagem e execução de testes em contêineres.

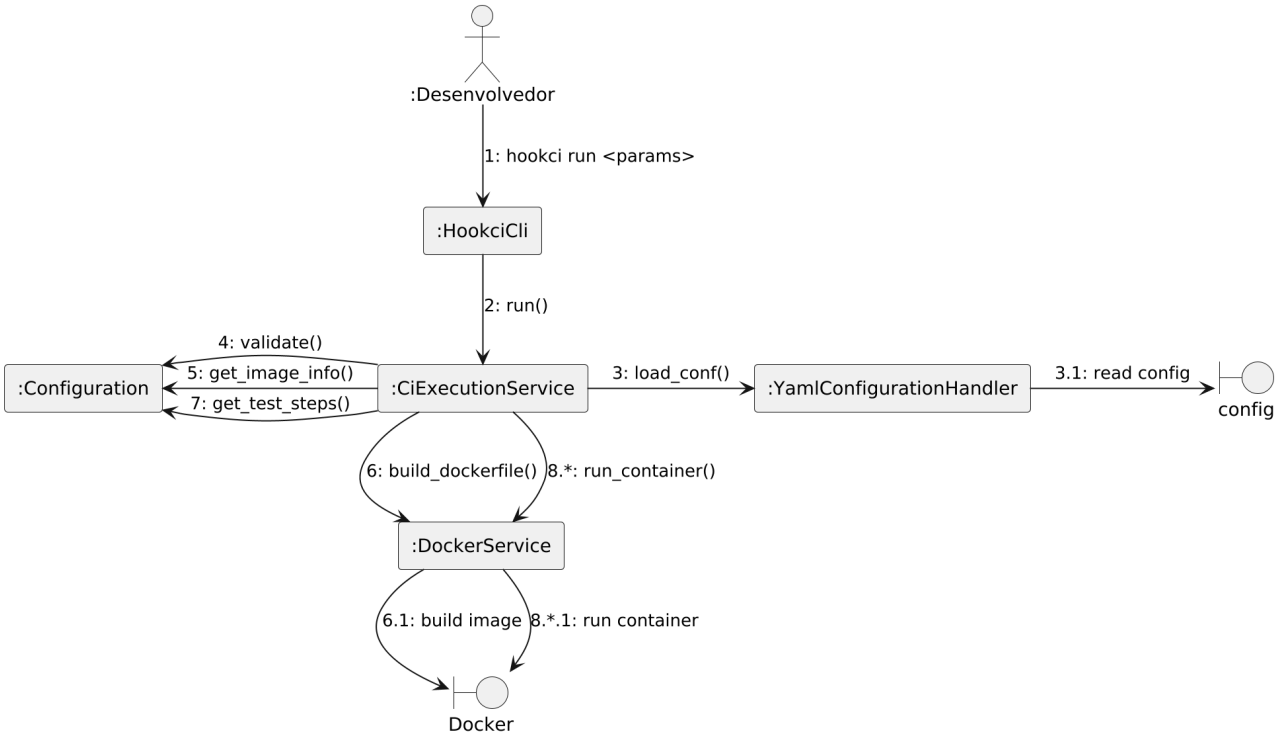

Figura 17. Diagrama de Comunicação para a operação "Executar CI"

A Figura 17 apresenta a comunicação para a execução manual da CI. A interação é iniciada pelo Desenvolvedor através do HookciCli. A sequência de colaborações entre CiExecutionService, YamlConfigurationHandler, Configuration, DockerService e as fronteiras ConfigFile e DockerSystem é semelhante à execução via *hook* (Figura 16).

3.4 Arquitetura

A arquitetura adota o padrão de camadas, visando a separação de responsabilidades e a manutenibilidade do código. A Figura 18 ilustra essa organização em quatro camadas principais: *Presentation*, *Application*, *Domain* e *Infrastructure*.

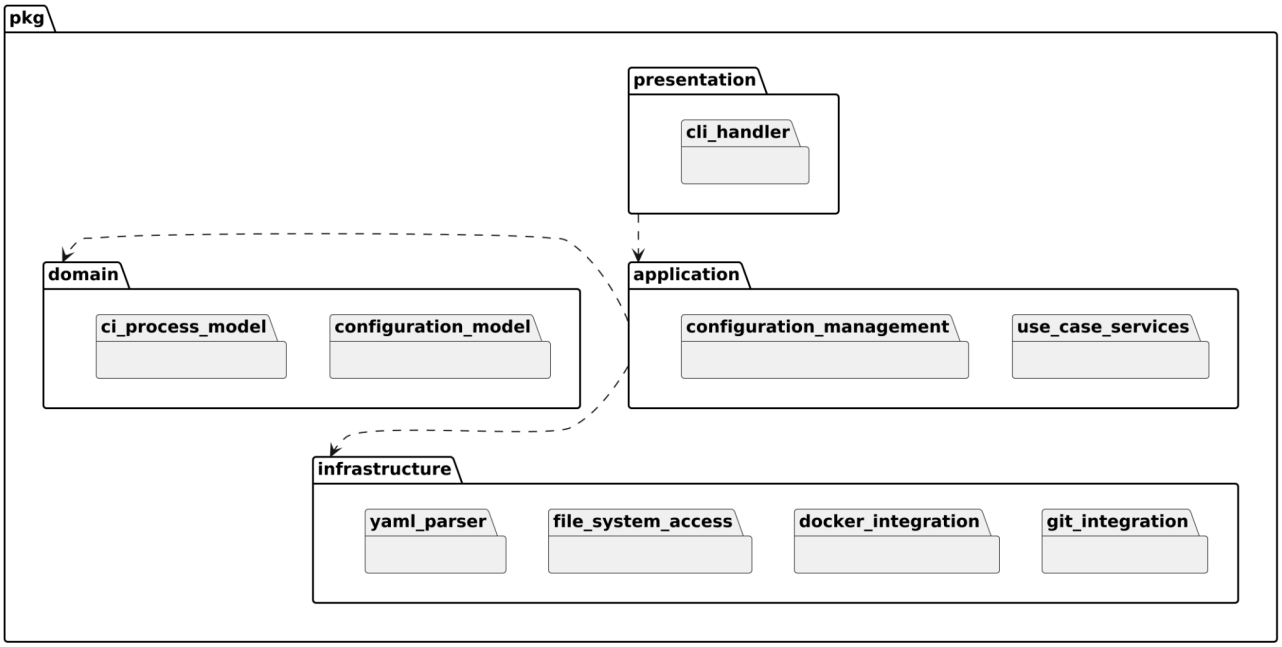

Figura 18. Diagrama de Pacotes da Arquitetura em Camadas

A camada de *Presentation* é responsável pela interação com o usuário, neste caso, através da CLI. Ela recebe os comandos do usuário e exibe os resultados. A camada de *Application* orquestra os casos de uso do sistema, coordenando as ações necessárias para atender às solicitações da camada de *Presentation*. Ela utiliza os serviços das camadas inferiores para executar suas tarefas, como gerenciamento de configuração e execução de fluxos de CI.

A camada de *Domain* encapsula a lógica de negócio central e as entidades do sistema, como os modelos de configuração e o processo de CI. Esta camada é independente das tecnologias externas e representa o núcleo do HookCI. Por fim, a camada de *Infrastructure* lida com detalhes técnicos e integrações externas, como a interação com Git, Docker, sistema de arquivos para leitura/escrita de configurações e a análise destas configurações. As dependências fluem das camadas superiores para as inferiores, garantindo que a lógica de negócio no *Domain* permaneça isolada das preocupações de infraestrutura e apresentação.

3.5 Diagramas de Estados

Para modelar o comportamento dinâmico do sistema durante a execução de um ciclo de CI, o Diagrama de Estados apresentado pela Figura 19 apresenta os principais estados e transições do processo de execução do HookCI.

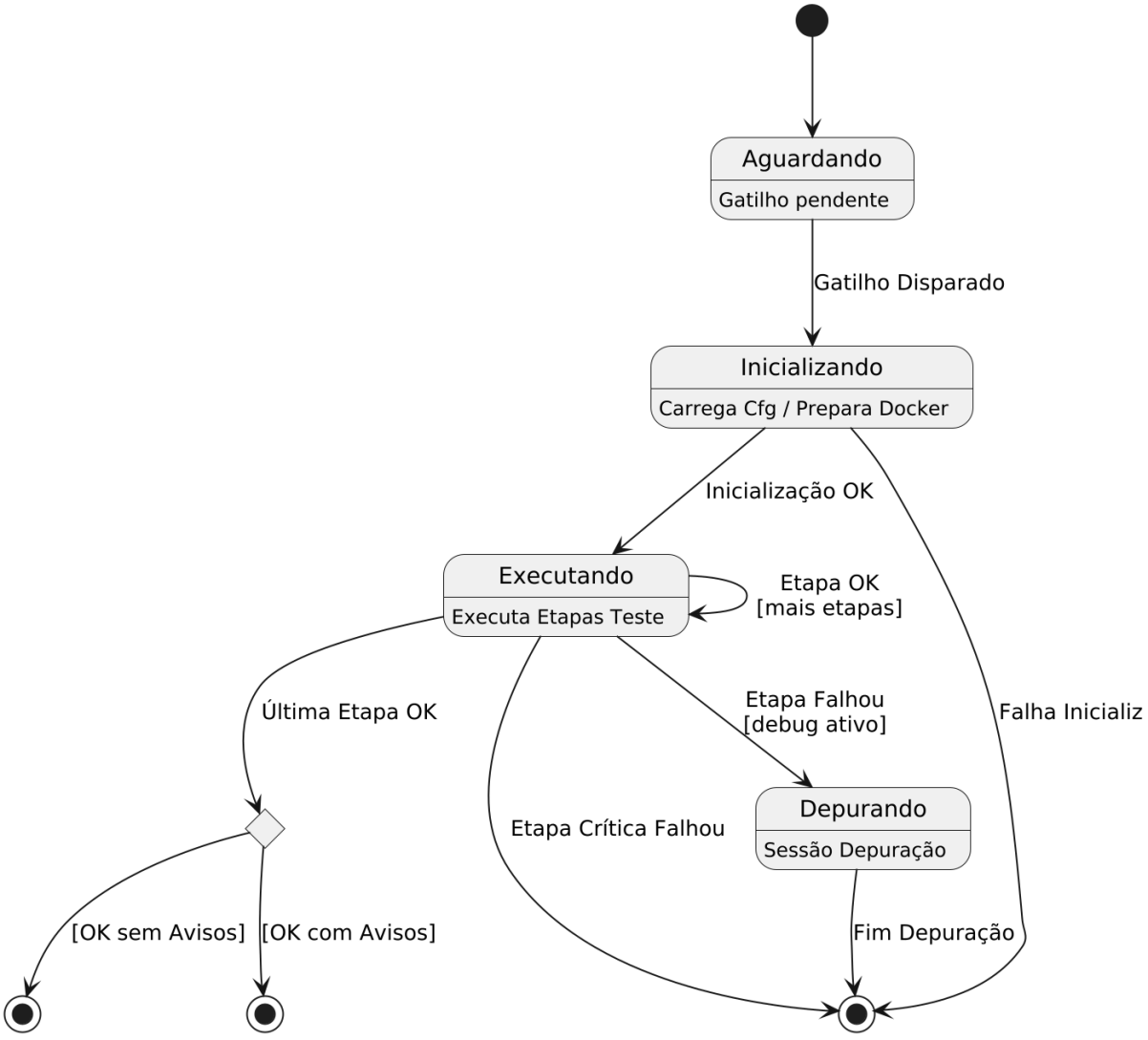

Figura 19. Diagrama de Estados da Execução de CI

O processo inicia no estado Aguardando, onde o sistema espera por um gatilho, seja um *hook* do Git ou um comando via CLI. Ao ser disparado, transita para o estado Inicializando, onde carrega a configuração e prepara o ambiente Docker. Se a inicialização for bem-sucedida, o sistema entra no estado Executando, processando sequencialmente cada etapa de teste definida na configuração.
Durante a execução, se uma etapa falhar e o modo de depuração estiver ativo, o sistema transitará para o estado Depurando. Caso contrário, ou após a depuração, se a etapa falha for crítica, o processo transita para o estado final Falha. Se uma etapa não crítica falhar, um aviso é gerado e a execução continua. Ao concluir todas as etapas, o sistema alcança um ponto de decisão que leva aos estados finais: OK, OK com Avisos ou Falha.

3.6 Diagrama de Componentes e Implantação.

Os diagramas de componentes e implantação detalham a estrutura modular do software e como ele é distribuído no ambiente de execução, respectivamente.
A Figura 20 exibe os principais componentes de software do HookCI e suas interconexões. O sistema é composto pelo componente de *Presentation*, *Application Services* que implementam os casos de uso, e o *Domain Model* com a lógica central. A interação com sistemas externos é mediada por adaptadores na camada de *Infrastructure*, como o "YAML/File Handler", o "Git Adapter" e o "Docker Adapter".

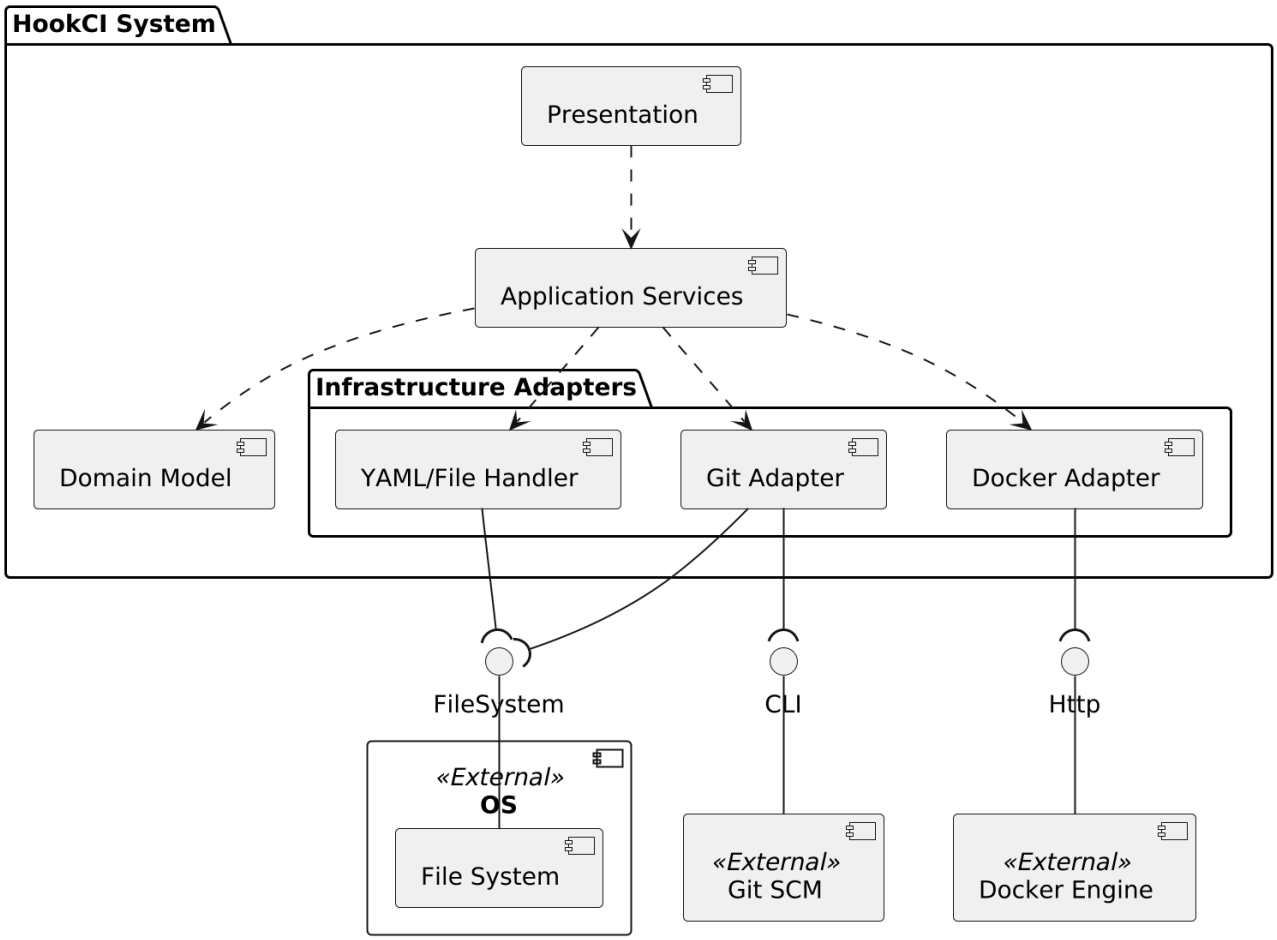

Figura 20. Diagrama de Componentes do Sistema HookCI

A Figura 21 ilustra o diagrama de implantação, mostrando como os artefatos de software são alocados nos nós do ambiente de execução típico, a estação de trabalho do desenvolvedor. Nesta estação residem o artefato principal HookCI CLI, o sistema de controle de versão Git, os *hooks* do Git configurados, o arquivo de configuração config.yaml e o código-fonte do projeto.

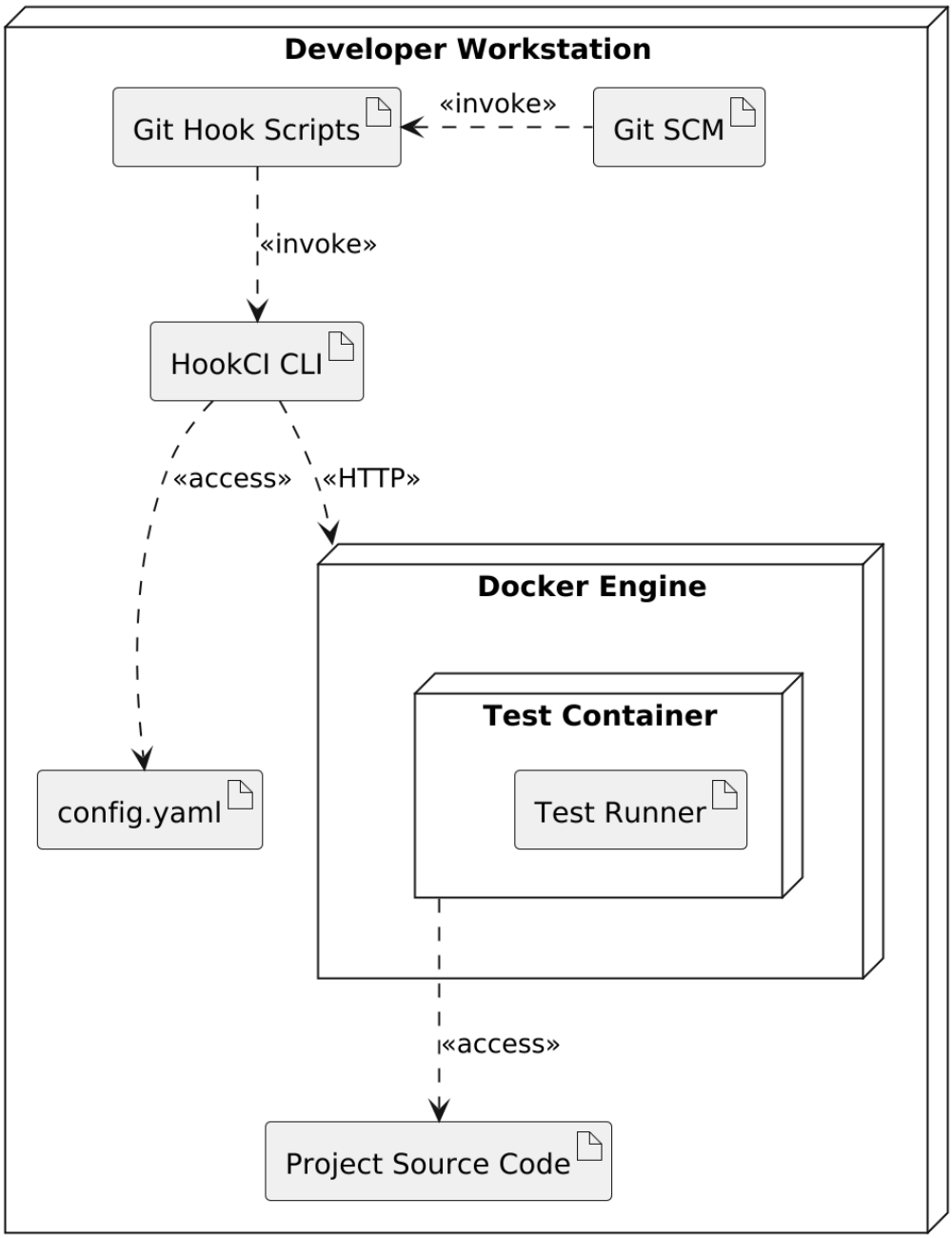

Figura 21. Diagrama de Implantação do Sistema HookCI

4. Projeto de Interface com Usuário

Esta seção apresenta as principais interações do usuário com o sistema HookCI por meio de sua CLI. Como o sistema opera exclusivamente via terminal, não são apresentados *mockups* ou *wireframes* gráficos. Em vez disso, são exibidas representações textuais das saídas geradas pelos comandos principais, ilustrando o fluxo de interação e o retorno fornecido ao usuário. Essas interações foram projetadas para cobrir as histórias de usuário definidas na Seção 2.3.

4.1 Esboço das Interfaces Comuns a Todos os Atores

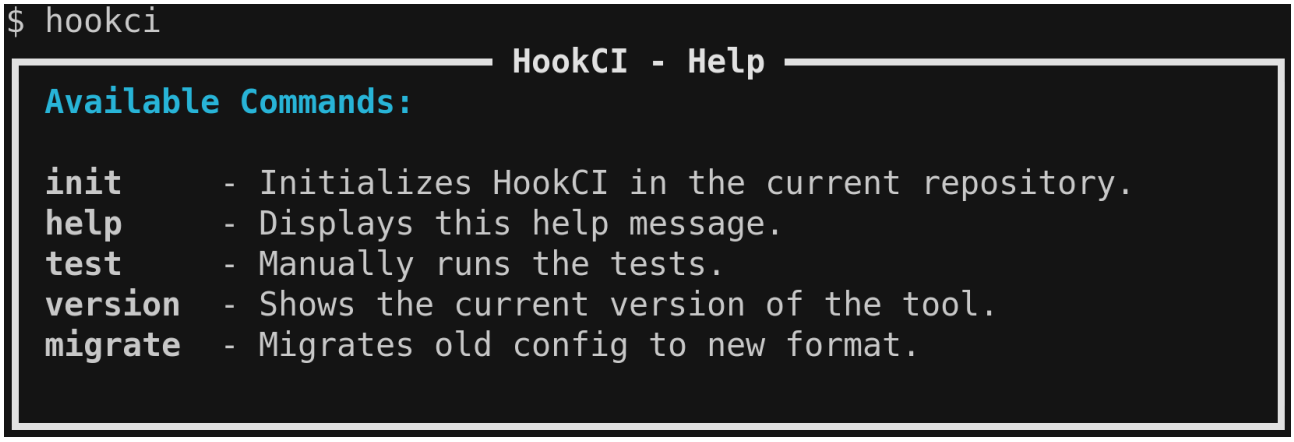

Figura 22. Descrição de comandos disponíveis

A Figura 22 exibe a saída do comando de ajuda, invocado quando o usuário executa a ferramenta sem argumentos ou com o comando *help*. A interface lista os comandos disponíveis e fornece uma breve descrição de cada um, auxiliando o usuário a entender as funcionalidades da ferramenta. Esta tela contempla diretamente a história de usuário US01.

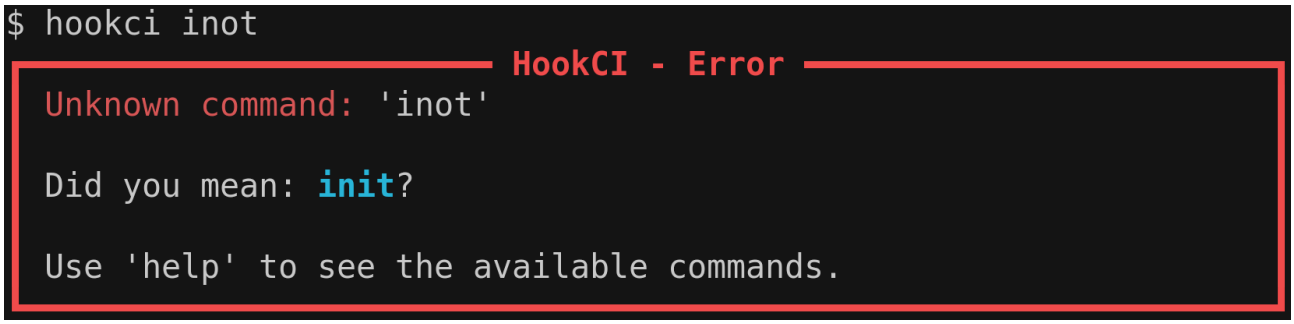

Figura 23. Comando inválido

A Figura 23 mostra a resposta do sistema quando um comando inválido é fornecido pelo usuário. A interface apresenta uma mensagem de erro, indicando qual comando foi considerado desconhecido e sugerindo o uso do comando *help* para visualizar as opções válidas. Embora não corresponda diretamente a uma história de usuário funcional principal, esta interface é crucial para a usabilidade e orientação do usuário, relacionando-se indiretamente com US01 ao direcionar para a ajuda.

4.2 Esboço das Interfaces Usadas pelo Desenvolvedor

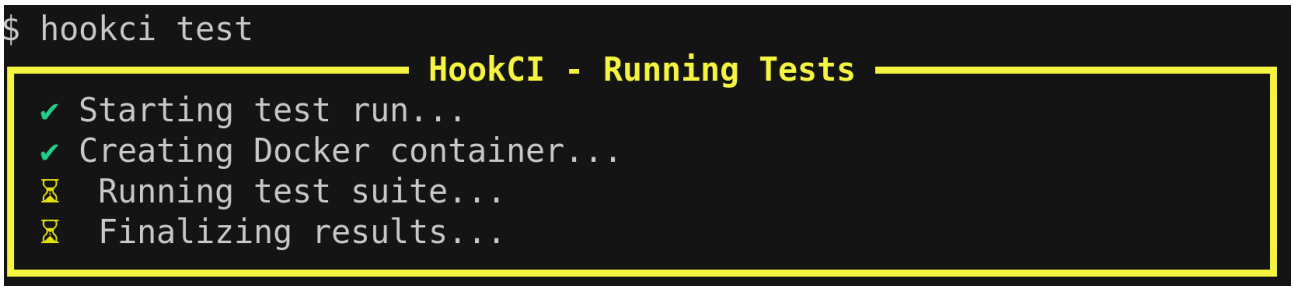

Figura 24. Execução de testes em andamento

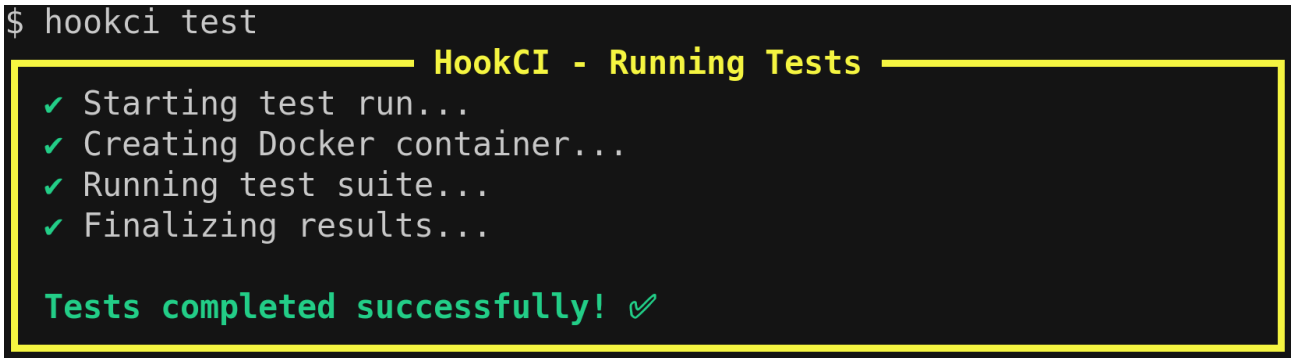

Figura 25. Execução de testes bem sucedida

A Figura 24 demonstra a saída da CLI durante a execução dos testes, acionada manualmente pelo comando *test* ou automaticamente por um Git *hook*. A interface exibe o progresso das etapas principais, como a inicialização, criação do contêiner Docker e execução da suíte de testes, utilizando indicadores visuais, sendo ⏳ para pendente/executando e ✔ para concluído. Assim pode-se fornecer retorno sobre o andamento. A Figura 25 ilustra a conclusão bem-sucedida da execução dos testes. Esta interação está relacionada às histórias de usuário US05, US09, US10, US11, US12, US13, US14, US15, US16, US17 e US18.

4.3 Esboço das Interfaces Usadas pelo Administrador

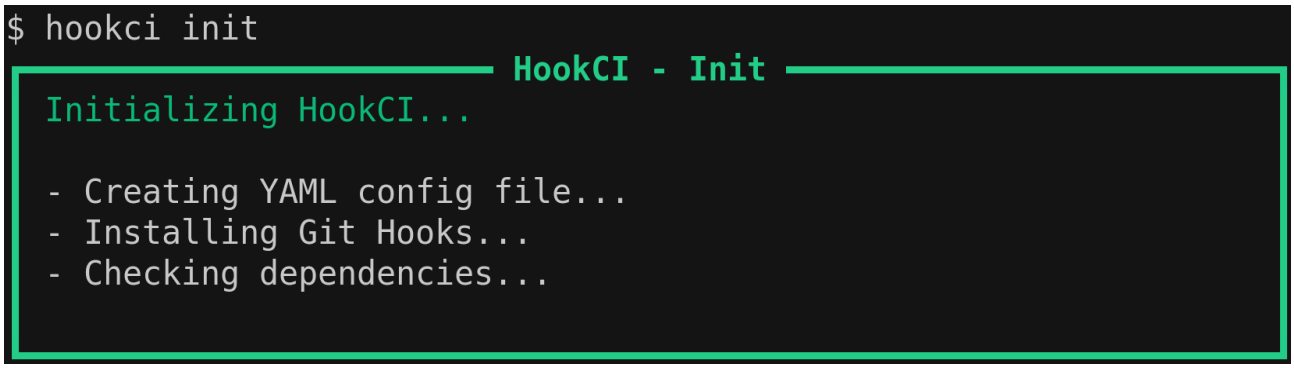

Figura 26. Inicialização de um repositório

A Figura 26 representa a saída do comando *init*, responsável por inicializar o HookCI em um repositório. A interface informa o início do processo e detalha as ações realizadas, como a criação

do arquivo de configuração YAML, a instalação dos Git *hooks* necessários e a verificação de dependências. Esta interação abrange as histórias de usuário US02, US03, US04.

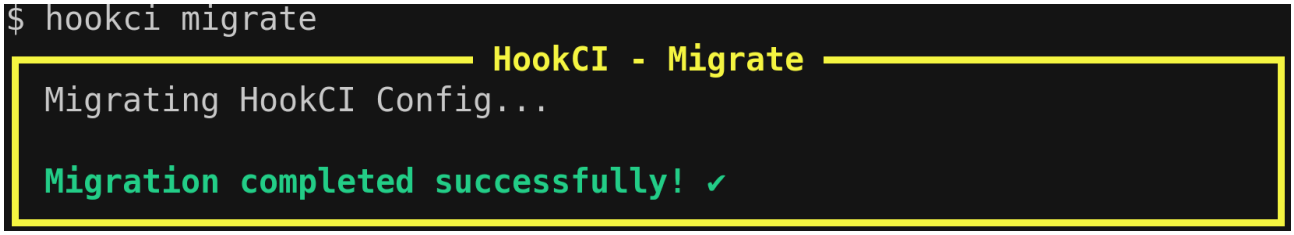

Figura 27. Migração de configuração bem sucedida

A Figura 27 demonstra a saída do comando de migração quando bem sucedido, executado manualmente pelo usuário para alterar sua configuração de uma versão anterior da ferramenta para a versão atual. Esta interação contempla as histórias de usuário US06 e US07.

5. Glossário e Modelos de Dados

Esta seção apresenta o glossário dos termos utilizados no arquivo de configuração YAML do HookCI e o modelo de dados que descreve sua estrutura. Como o sistema não utiliza um banco de dados tradicional, o foco está na representação dos dados de configuração, que são armazenados e lidos a partir de um arquivo YAML. A tabela a seguir detalha os atributos de configuração.

Atributo	**Formato**	**Descrição**
version	string	Versão do esquema de configuração do HookCI. Utilizada para o gerenciamento de migrações entre diferentes versões da ferramenta.
log_level	string	Define o nível de verbosidade dos logs da CLI.
docker	object	Agrupa as configurações relacionadas ao ambiente Docker utilizado para a execução dos testes.
image	string	Nome e tag da imagem Docker pré-existente a ser utilizada. Se omitido, “dockerfile” deve ser especificado.
dockerfile	string	Caminho relativo para um Dockerfile no repositório, que será usado para construir a imagem Docker. Se omitido, “image_name” deve ser usado.
hooks	object	Define quais *hooks* do Git serão gerenciados e ativados pelo HookCI.
pre-commit	boolean	Se verdadeiro, ativa a execução do HookCI automaticamente durante o evento *pre-commit* do Git.

pre-push	boolean	Se verdadeiro, ativa a execução do HookCI automaticamente durante o evento *pre-push* do Git.
steps	objects	Etapas a serem executadas sequencialmente durante o processo de CI.
name	string	Nome descritivo e único para a etapa de teste, utilizado para identificação nos logs.
command	string	O comando a ser executado dentro do contêiner Docker para uma etapa.
critical	boolean	Se verdadeiro (padrão), uma falha nesta etapa interrompe a execução da CI e aborta a operação Git. Se falso, uma falha gera apenas um aviso.
env	object	Um mapa de pares chave-valor representando variáveis de ambiente a serem injetadas no contêiner Docker para uma etapa.
VAR	string	Valor de variável de ambiente a ser definida para uma etapa.

Tabela 8. Glossário do Arquivo de Configuração YAML

A Figura 28 a seguir ilustra, a estrutura hierárquica e os tipos de dados esperados para a configuração explicada acima, servindo como um esquema para sua validação e interpretação pela ferramenta.

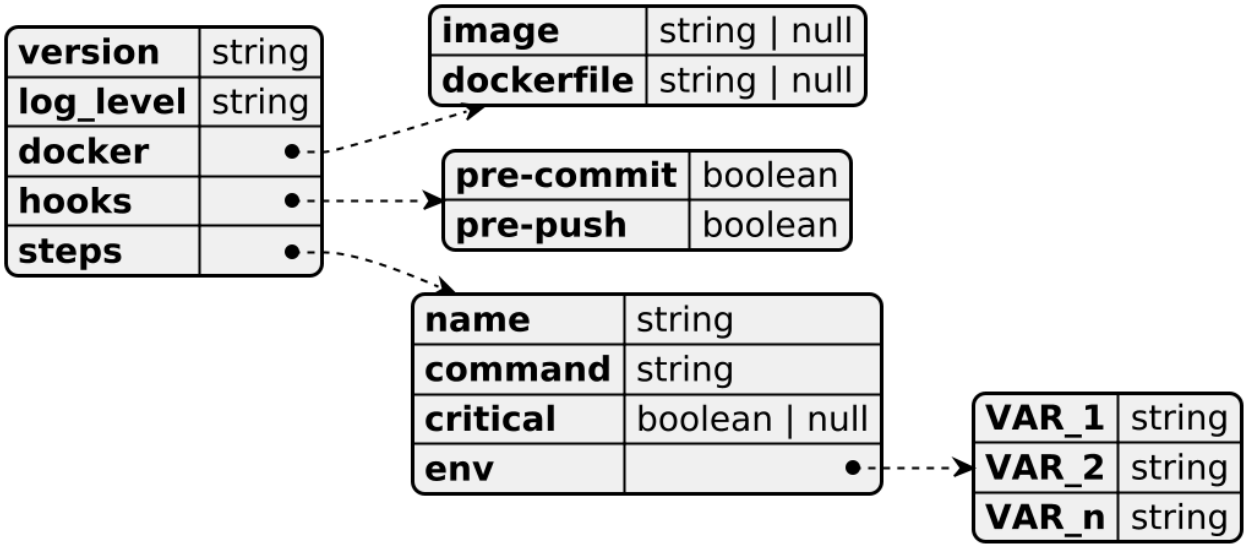

Figura 28. Mapeamento da Estrutura do Arquivo de Configuração YAML

6. Casos de Teste

Uma descrição de casos de teste para validação do sistema.

7. Cronograma e Processo de Implementação

Uma descrição do cronograma para implementação do sistema e do processo que será seguido durante a implementação.