Automação de Deploy e Padronização de Ambientes Web: Um Estudo de Caso Utilizando Docker Compose e GitHub Actions.

Maria Clara Nunes Linhares[1], Raynara Gustavo da Costa[1], Lucas Denner Hilláryo da Silva[1], Paulo Eugenio da Costa Filho[1], Diego da Silva Pereira[1]

[1] Laboratório de Pesquisa em Redes e Sistemas Computacionais (NOCS LAB), Instituto Federal do Rio Grande do Norte (IFRN), Parnamirim, Brasil
(maria.linhares@escolar.ifrn.edu.br)

Resumo: Este trabalho apresenta a implementação de um testbed para avaliar a automação de deploy e a padronização de ambientes web usando contêineres. Diante das falhas geradas por divergências de configuração entre ambientes, utilizou-se o Docker Compose para empacotar uma aplicação multicontêiner composta por uma API em Flask e um banco de dados MySQL. A automação do fluxo de entrega foi estruturada via GitHub Actions. Os resultados validaram a eliminação de erros manuais e a consistência da infraestrutura.

Palavras-chave: Docker Compose; DevOps; GitHub Actions; Automação de Deploy; Flask.

INTRODUÇÃO

O desenvolvimento de aplicações web tornou-se um dos pilares da tecnologia da informação contemporânea Essas aplicações oferecem grande facilidade de distribuição e atualização quando comparadas a softwares tradicionais instalados localmente. Conforme destacado por Pressman e Maxim (2020), as aplicações web representam uma das formas mais eficientes de entrega de software em uma sociedade cada vez mais conectada e orientada a serviços digitais.

Entretanto, a implantação desses sistemas impõe desafios técnicos, destacando-se as limitações relacionadas à padronização entre os ambientes de desenvolvimento, testes e produção. Diferenças em sistemas operacionais e versões de bibliotecas podem ocasionar comportamentos inesperados durante as implantações, comprometendo a estabilidade do sistema. A cultura DevOps surge como uma resposta a essas demandas ao propor a automação de processos e a redução de intervenções manuais ao longo do ciclo de vida do software (IBM, 2023).

Neste cenário de padronização, a conteinerização passou a ser amplamente utilizada. Esta tecnologia consiste no isolamento de ambientes, encapsulando todas as dependências necessárias para o funcionamento das aplicações (Miell e Sayers, 2016). Para gerenciar múltiplos contêineres interdependentes de forma simplificada em um único nó, a ferramenta Docker Compose se destaca por permitir a definição e execução de pilhas de serviços integrados, como o framework Flask (Python) e o sistema gerenciador de banco de dados MySQL, a partir de um único arquivo descritivo YAML.

Diante do contexto apresentado, este trabalho tem como objetivo geral implementar e validar um testbed automatizado focado na padronização e no deploy contínuo de uma aplicação multicontêiner (Flask e MySQL). Busca-se avaliar a viabilidade prática da ferramenta Docker Compose combinada ao agente de automação GitHub Actions para eliminar inconsistências de ambiente e acelerar o fluxo de entrega de software.

MATERIAL E MÉTODOS

As metodologias utilizadas no trabalho para atingir os objetivos propostos basearam-se no desenvolvimento técnico e na validação de um testbed composto por uma aplicação multicontêiner. A metodologia utilizada nesta pesquisa teve como base a exploração da ferramenta Docker associada ao utilitário Docker Compose e à plataforma GitHub Actions. O ecossistema Docker é amplamente usado em ambientes acadêmicos e profissionais no ensino de infraestrutura de sistemas, cultura DevOps e temas correlatos. Sua interface de linha de comando e arquivos descritivos são intuitivos e estimulam os estudantes na busca de implementação e automação dos mais diversos cenários de implantação.

A ferramenta foi utilizada para estruturar um sistema de agendamento de tarefas desenvolvido em Python com o microframework Flask, integrado a um banco de dados relacional MySQL. O ambiente foi planejado para mitigar problemas clássicos de divergência de configuração entre ambientes computacionais, utilizando volumes do Docker para garantir a persistência e segurança das informações.

O fluxo metodológico e a validação do aprendizado prático foram divididos no cumprimento de etapas sequenciais: a escrita do código em Flask, a configuração do isolamento da aplicação através de um Dockerfile, a centralização dos serviços no arquivo docker-compose.yml (com redes virtuais isoladas do tipo bridge) e o desenvolvimento do fluxo de automação (workflow) no GitHub Actions para compilação e publicação automatizada de imagens no Docker Hub. É importante destacar que esse cenário e sua estrutura de arquivos foram aplicados para validar de forma empírica a consistência e a viabilidade da automação do deploy.

RESULTADOS E DISCUSSÃO

A execução prática do testbed comprovou a eficiência da utilização do Docker Compose e da cultura DevOps na padronização de ambientes computacionais. A inicialização da pilha de serviços foi realizada no terminal local, conforme apresentado na Figura 1, por meio do comando d*ocker compose up --build*, o qual acionou a criação da rede customizada docker-project_app-net e o provisionamento dos contêineres de banco de dados (db-1) e da aplicação web (web-1).

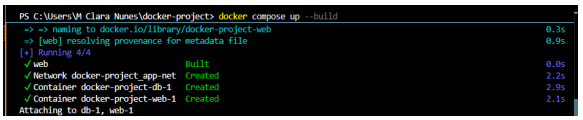

Figura 1. Inicialização e criação da infraestrutura multicontêiner via terminal.

A interface do Agendador de Tarefas desenvolvido pôde ser acessada localmente através do navegador, exibindo uma página inicial amigável (Figura 2). O sistema dispõe de rotas funcionais de CRUD que permitem aos usuários interagir diretamente com a base de dados MySQL.

Figura 2. Tela inicial do sistema Agendador de Tarefas via navegador.

As funcionalidades da aplicação englobam a listagem dinâmica de tarefas recuperadas do banco de dados (Figura 3) e um formulário específico para a inserção de novos registros (Figura 4), consolidando o fluxo completo de persistência de dados em ambiente conteinerizado.

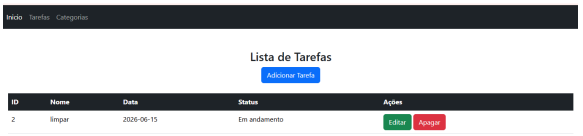

Figura 3. Visualização e listagem das tarefas cadastradas na aplicação.

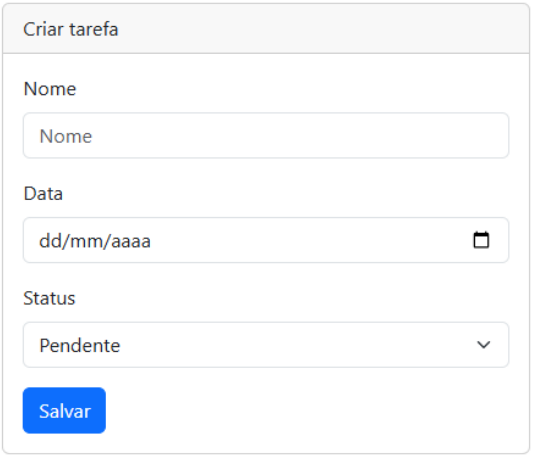

Figura 4. Formulário de inserção para cadastro de novas tarefas no sistema.

No que tange à automação da infraestrutura, o pipeline de CI/CD foi validado com êxito utilizando o GitHub Actions. Para garantir a segurança nas etapas de publicação, foram configuradas chaves secretas criptografadas diretamente nas configurações do repositório, denominadas DOCKER_USERNAME e DOCKER_PASSWORD, conforme ilustrado na Figura 5.

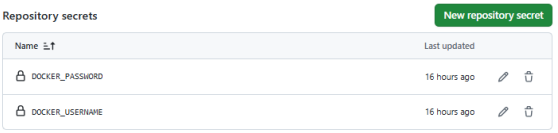

Figura 5. Configuração de segredos no repositório GitHub para autenticação.

A partir de cada modificação disparada via git push, o fluxo definido no arquivo *ci-cd.yml* é executado pelo agente. A Figura 6 exibe o sumário com o sucesso das duas etapas principais do pipeline: o estágio de compilação/teste (build-and-test) e o estágio subsequente de entrega (publish-image), totalizando um ciclo de 1 minuto e 49 segundos. É importante ressaltar que o testbed foi realizado em notebook pessoal simples.

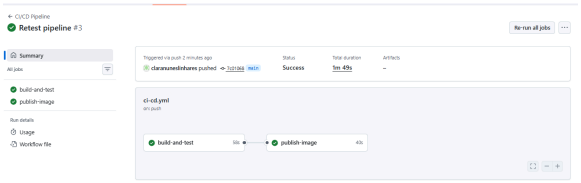

Figura 6. Sumário de execução com sucesso do fluxo automatizado de CI/CD.

O detalhamento do estágio de publicação da imagem é mapeado na Figura 7, evidenciando as subetapas sequenciais executadas pelo executor virtual do GitHub, que compreendem desde o login automatizado na plataforma Docker Hub até o encerramento do job.

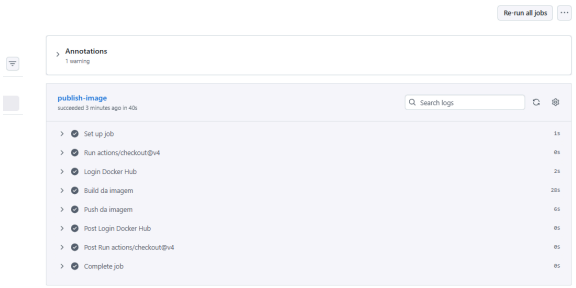

Figura 7. Logs detalhados das etapas de compilação e publicação da imagem.

A imagem consolidada e tagueada como estável (*latest*) foi disponibilizada no repositório público do Docker Hub. Na máquina de implantação, realizou-se o download atualizado dessa nova versão gerada pelo pipeline através do comando docker pull, conforme validado no terminal da Figura 8.

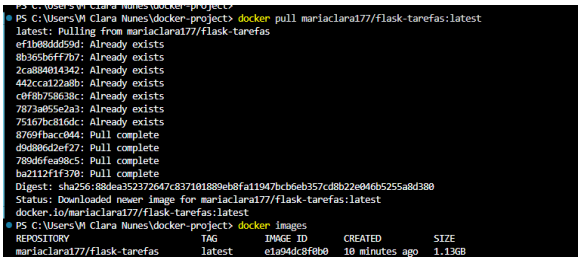

Figura 8. Download da imagem gerada pelo pipeline e listagem das imagens locais.

Após a conclusão do pull da nova versão do sistema, o ambiente de produção foi atualizado de forma ágil utilizando o Docker Compose em segundo plano (modo detached). A Figura 9 comprova a reinicialização bem-sucedida e o status ativo dos contêineres (db-1 e web-1) rodando a versão recém-atualizada da aplicação.

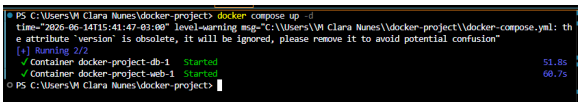

Figura 9. Inicialização e verificação de status dos contêineres ativos após o pull da nova imagem.

CONCLUSÃO

O presente estudo permitiu validar a viabilidade e a eficácia da utilização da tecnologia de conteinerização com o Docker Compose associada a pipelines de CI/CD via GitHub Actions. Diante dos testes práticos executados, conclui-se que a centralização da infraestrutura em um arquivo descritivo único eliminou com sucesso as inconsistências geradas por divergências de configuração entre ambientes, garantindo a padronização e a portabilidade do sistema de agendamento de tarefas.

A automação do fluxo de entrega contínua mostrou-se altamente eficiente. O pipeline reduziu drasticamente o tempo de disponibilização de novas versões do software e mitigou a ocorrência de erros humanos decorrentes de processos de implantação manuais. Ademais, a resolução dos problemas de dependência de inicialização entre a aplicação Flask e o SGBD MySQL através das diretivas do Compose provou a resiliência local do ambiente configurado.

Como limitação da pesquisa, aponta-se que o Docker Compose opera de forma centralizada em cenários de nó único (single-node), não cobrindo requisitos avançados de alta disponibilidade geográfica e balanceamento de carga distribuído nativo. Como sugestão para trabalhos futuros, recomenda-se a expansão deste modelo para orquestradores de clusters, como o Kubernetes ou o Docker Swarm, além da inclusão de ferramentas de monitoramento em tempo real para uma gestão ainda mais proativa da infraestrutura.

AGRADECIMENTOS

Os autores agradecem ao apoio do NOCS LAb e ao IFRN Parnamirim.

REFERÊNCIAS

GRINBERG, Miguel. Flask Web Development: developing web applications with Python. 2. ed. O'Reilly Media, Sebastopol, 2018.

IBM. O que é DevOps? Think Topics, IBM Brasil, 2023.

MIELL, Ian; SAYERS, Aidan Hobson. Docker in Practice. 1. ed. Manning Publications, Shelter Island, NY, 2016.

MILANI, André. MySQL: guia do programador. 1. ed. Novatec Editora, São Paulo, 2008.

PRESSMAN, Roger S.; MAXIM, Bruce R. Engenharia de software: uma abordagem profissional. 9. ed. AMGH, Porto Alegre, 2020.

RED HAT. O que é CI/CD? Red Hat Topics, Red Hat Brasil, 2022.