BANNER DO EVENTO

Docker Multistage na prática

NOBRE JUNIOR, A. X.[1]; TELES, R. M.[2]
1. Estudante do curso de bacharelado em Sistemas de Informação, IF Goiano – Campus Ceres.
2. Professor orientador, IF Goiano – Campus Ceres.

INTRODUÇÃO

Durante o desenvolvimento de projetos de aplicações é muito comum que se encontre problemas e conflitos de versões de aplicações e sistemas operacionais dos diferentes desenvolvedores envolvidos na construção do ambiente, durante o desenvolvimento do projeto do SLab que é um software de ferramenta estatística, se enfrentou esse problema, isso causou o surgimento de alternativas conhecidas como Conteinerização que é muito comum sua utilização por serem consideradas leves, rápidas e balanceadas, a ferramenta mais famosa e para pronto uso é o *Docker*, esse Docker serve para fazer essa separação de recursos utilizados no desenvolvimento e revisando a documentação, os desenvolvedores identificaram uma opção de usar diferentes ferramentas em apenas um arquivo docker, diminuindo, em teoria, o tamanho da imagem de base que esse *container* vai usar. Deve-se observar que essa prática vai de fato refletir e se a mesma poderá ser construída de diferentes formas.

MATERIAIS E MÉTODOS

Usando as recomendações da documentação do docker buscou-se entender o conceito de *docker multistage* para implementá-lo no projeto, durante a criação da imagem que o *container* vai usar como base para rodar a aplicação, poderemos observar a diferença de tamanhos geradas na criação com o *docker multistage* e a imagem gerada em um pequeno servidor Ubuntu 22.04, adicionando as dependências das aplicações. Para isso vai se precisar de um computador/notebook com capacidade de virtualização, conhecida como *hypervisor*, com *docker* na versão 24 instalado, será criado os aquivos para criação das máquinas e com as respectivas configurações para cada ferramenta, após isso será realizado o comando de execução do *docker* para cada um dos arquivos e comparado o tamanho final de cada uma das imagens geradas com as dependências preenchidas, nesse caso usamos NodeJS na versão 18, PHP na versão 8.1 e Composer na versão 2.5.

RESULTADOS

Como resultados foi observado que a utilização do *multistage* dentro de projetos gerou uma imagem *bem menor*, diferente do que foi executado separado onde as dependências eram criadas e mantidas dentro do repositório local de imagens, isso abre porta para uma diminuição dos recursos de armazenamentos em *Runners* de *pipelines* promovendo a redução de valores de armazenamentos em projetos reais que no geral usam esses *runners* para manter.

CONSIDERAÇÕES FINAIS

Concluindo podemos promover a divulgação do recurso do docker promovendo a visibilidade e redução de custos de armazenameto, abrindo possibilidades para outras otimizações dentro do ambiente docker. Pontos Importantes para serem considerados:

- *Docker* consegue manter-se atualizada frente as necessidades do mercado;
- Foi possível otimizar os recursos dos *runners* do projeto que usariam maiores recursos para construir uma nova imagem da aplicação;

REFERÊNCIAS BIBLIOGRÁFICAS

https://docs.docker.com/