Instituto Federal de Educação, Ciência e Tecnologia de São Paulo

Campus São João da Boa Vista

Implementação e Configuração de um Servidor Web

Abordagem Prática com Docker Compose, NGINX e Aplicações Open Source

Autores:

Fernanda Martins da Silva
Gabriel Maia Miguel
Samuel Oliveira Lopes

Relatório Técnico

Unidade Curricular: Redes de Computadores

Novembro de 2025

Resumo

Este relatório detalha o processo de implementação e configuração de uma infraestrutura de servidores web baseada em microsserviços, utilizando tecnologias de código aberto (*open source*) orquestradas via Docker. O projeto visa demonstrar a criação de um ambiente robusto e escalável, integrando Sistemas Gerenciadores de Banco de Dados (SGBD) heterogêneos (MariaDB 10.6 e PostgreSQL 15), um servidor *gateway* e proxy reverso (NGINX), e aplicações web com pilhas tecnológicas distintas (Moodle 4.5 em PHP e Spring Petclinic em Java 25). A metodologia adotada privilegia a conteinerização com Docker Compose para assegurar a modularidade, o isolamento de redes e a portabilidade do ambiente. São apresentados os arquivos de configuração, as estratégias de otimização de imagens (*multi-stage builds*) e a resolução de desafios relacionados ao roteamento de tráfego e persistência de dados.

Palavras-chave: Servidor Web, NGINX, Docker Compose, PostgreSQL, Moodle, Spring Boot, Microsserviços.

Sumário

1 Introdução e Objetivos

A ubiquidade das aplicações web modernas exige infraestruturas que sejam não apenas funcionais, mas também seguras, modulares e de fácil manutenção. Este projeto insere-se no contexto prático da disciplina de Laboratório de Redes de Computadores, visando consolidar conhecimentos sobre protocolos HTTP, gestão de bases de dados e orquestração de serviços.

1.1 Objetivo Geral

O objetivo primordial é demonstrar a implementação técnica de um servidor web capaz de hospedar múltiplas aplicações concorrentes, com diferentes requisitos de execução (*runtime*), em um único ambiente coeso e acessível via rede local (LAN).

1.2 Objetivos Específicos

Para atingir o objetivo geral, a equipe definiu os seguintes passos:

1. **Gestão de Dados:** Implementação e configuração de SGBDs relacionais distintos (MariaDB e PostgreSQL) em contêineres isolados.
2. **Aplicações Heterogêneas:**
    - Hospedagem de uma aplicação Java Web (Spring Petclinic) utilizando as versões mais recentes do Java (v25).
    - Configuração de uma plataforma LMS (*Learning Management System*) baseada em PHP (Moodle 4.5).
3. **Roteamento de Tráfego:** Configuração de um servidor NGINX atuando como Proxy Reverso para gerenciar as requisições na porta 8080 e distribuí-las pelos serviços internos.
4. **Documentação:** Elaboração técnica dos artefatos de configuração (`Dockerfile`, `compose.yml`, `nginx.conf`).

2 Fundamentação Teórica e Arquitetura

A solução foi desenhada seguindo o padrão de arquitetura de microsserviços, onde cada função do sistema é encapsulada na sua própria unidade de execução (contêiner).

2.1 Tecnologias Adotadas

A escolha das tecnologias baseou-se na estabilidade, suporte da comunidade e conformidade com padrões *open source*. A Tabela 1 resume a pilha tecnológica extraída dos arquivos de configuração.

Tabela 1: Versões e Tecnologias Utilizadas

Serviço	Tecnologia/Imagem	Papel no Sistema
Web Gateway	`nginx:latest`	Proxy Reverso e Estáticos
LMS App	`brandkern/moodle:4.5-latest`	Plataforma de Ensino
LMS DB	`mariadb:10.6`	Persistência para o Moodle
Java App	`eclipse-temurin:25` (Custom)	Aplicação Spring Petclinic
Java DB	`postgres:15-alpine`	Persistência para o Petclinic

2.2 Topologia de Rede e Isolamento

Um aspecto crítico da segurança em Docker é o isolamento de redes. Em vez de utilizar a rede padrão (*default bridge*), foram criadas duas redes internas distintas para garantir que as bases de dados sejam acessíveis apenas pelas suas respectivas aplicações:

- **`backend`**: Rede dedicada à pilha Java. Conecta exclusivamente o NGINX, a aplicação Petclinic e a base de dados PostgreSQL.
- **`moodle_net`**: Rede dedicada à pilha PHP. Conecta exclusivamente o NGINX, o Moodle e a base de dados MariaDB.

O contêiner `web` (NGINX) é o único elemento da arquitetura conectado a ambas as redes, atuando como uma ponte de aplicação (*Application Layer Gateway*) segura entre o mundo externo e os serviços internos.

3 Implementação Detalhada

3.1 Roteamento (NGINX)

O NGINX foi configurado para ser o único ponto de contato com o mundo exterior, escutando na porta **8080**. A sua função de Proxy Reverso é vital para permitir que múltiplas aplicações compartilhem o mesmo endereço IP e porta. Além disso, foi configurado um volume específico para servir a documentação estática do projeto:

```
./web/public:/usr/share/nginx/html
```

Esta diretiva mapeia os arquivos locais da equipe para o diretório padrão de publicação do servidor web.

4 Análise Detalhada dos Artefatos de Configuração

Nesta seção, a equipe apresenta a análise técnica dos arquivos de configuração desenvolvidos, explicando a função de cada diretiva.

4.1 Orquestração: `compose.yml`

Este arquivo define a infraestrutura como código (IaC).

- **Serviços (`services`):**
  - `web`: Utiliza a imagem `nginx:latest`. A porta 8080 do host é mapeada para a 80 do contêiner. A dependência `depends_on` garante que o proxy inicie apenas após as aplicações estarem prontas.
  - `petclinic`: Construído a partir de um `Dockerfile` local (contexto `./spring-petclinic`). Utiliza a variável de ambiente `SPRING_PROFILES_ACTIVE=postgres` definida no arquivo externo `config.env` para ativar o driver JDBC correto.
- **Volumes (`volumes`):**
  - Volumes nomeados como `moodle_db_data` e `petclinic_db_data` garantem a persistência dos dados. O Docker gerencia o local de armazenamento no host, protegendo os dados contra a exclusão acidental dos contêineres.

4.2 Construção da Aplicação Java: `Dockerfile`

O arquivo de construção do `petclinic` é um exemplo clássico e eficiente do padrão *Multi-stage Build*. Esta técnica divide o processo de criação da imagem em fases distintas para otimizar o tamanho e a segurança do artefato final.

```
# Estágio 1: Build
FROM eclipse-temurin:25-jdk AS build
WORKDIR /workspace
COPY . .
RUN ./mvnw package -DskipTests

# Estágio 2: Runtime
FROM eclipse-temurin:25-jre
WORKDIR /app
COPY --from=build /workspace/target/*.jar app.jar
ENTRYPOINT ["sh", "-c", "java $JAVA_OPTS -Dserver.servlet.context-path=/
    petclinic -jar /app/app.jar"]
```

Listing 1: Dockerfile com Multi-stage Build

Explicação Técnica:

1. **Estágio de Compilação (`AS build`):** Inicia-se com uma imagem base completa (JDK 25), que contém compiladores e ferramentas de desenvolvimento. O código fonte é copiado e o Maven Wrapper (`mvnw`) é executado para compilar a aplicação e gerar o arquivo `.jar`. Note que este estágio é pesado, contendo código fonte e dependências de build.
2. **Estágio de Execução (`Runtime`):** Inicia-se um novo estágio com uma imagem base minimalista (JRE 25), que contém apenas o ambiente de execução Java, sem compiladores.

3. **Transferência de Artefato (`COPY --from=build`):** Apenas o arquivo compilado (`.jar`) é copiado do estágio anterior para a nova imagem. Todo o código fonte, cache do Maven e ferramentas de desenvolvimento são descartados.

4. **Resultado:** Obtém-se uma imagem final significativamente menor e mais segura para produção, pois reduz a superfície de ataque ao não conter ferramentas desnecessárias.

4.3 Configuração do Proxy: `nginx.conf`

O arquivo define o comportamento do servidor de borda, com estratégias distintas para cada aplicação:

- **Rota Petclinic:** A diretiva `proxy_pass http://petclinic:8080;` encaminha o tráfego para o serviço Java. O uso do nome do serviço (`petclinic`) é resolvido pelo DNS interno do Docker.

- **Rota Moodle:** A diretiva `proxy_pass http://moodle;` foi configurada **sem a barra final** (*trailing slash*).

    - *Justificativa Técnica:* Conforme observado nos arquivos de configuração, omitir a barra instrui o NGINX a anexar a URI original da requisição ao encaminhá-la para o upstream. Isso garante que requisições como `/moodle/login/index.php` sejam recebidas corretamente pelo container Apache interno.

- **Headers de Transparência:**

    - `proxy_set_header Host $http_host`: Preserva o cabeçalho original, essencial para que a aplicação de backend saiba qual domínio o cliente acessou.
    - `X-Forwarded-For`: Adiciona o IP real do cliente à requisição, permitindo logs de auditoria corretos e controle de acesso baseado em IP na aplicação final.

4.4 Variáveis de Ambiente: `config.env`

Aderindo aos princípios da metodologia *12-Factor App*, as configurações sensíveis foram externalizadas.

- `POSTGRES_URL`: Define a string de conexão JDBC, apontando para o hostname `petclinic-db`.

- `WWWROOT`: Configuração crítica para o Moodle, informando-o de que ele está sendo acessado via `http://localhost:8080/moodle`, permitindo a geração correta de URLs absolutas para recursos estáticos (CSS/JS).

5 Desafios e Soluções

Durante a implementação, a equipe deparou-se com desafios técnicos, especificamente relacionados ao *Context Path*.

Problema: As aplicações assumem nativamente a execução na raiz (`/`). Ao servi-las em subdiretórios (`/moodle` e `/petclinic`), links internos quebravam.

Solução: Reconfiguração explícita:

- **Java:** Adição da propriedade `-Dserver.servlet.context-path=/petclinic` no `ENTRYPOINT` do Dockerfile.
- **Moodle:** Definição correta da variável `WWWROOT` no arquivo de ambiente, alinhada com a configuração do bloco `location` do NGINX.

6 Conclusão e Trabalhos Futuros

A infraestrutura implementada cumpriu integralmente os requisitos propostos. A orquestração via Docker Compose garantiu um ambiente reprodutível e modular. A segmentação de redes demonstrou-se eficaz para a segurança, e a estratégia de *multi-stage build* no Dockerfile da aplicação Petclinic assegurou uma entrega de software otimizada.

Como trabalhos futuros, sugere-se a implementação de certificados SSL/TLS (HTTPS) para garantir a confidencialidade dos dados em trânsito.

Repositório do Projeto: https://github.com/sunref/redc