FACULDADE DE INFORMÁTICA E ADMINISTRAÇÃO PAULISTA TECNOLOGIA DE ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

1º CP-2º Semestre –Docker Compose

ARTHUR BISPO DE LIMA RM: 557568 - 2TDSPV

JOÃO PAULO MOREIRA DOS SANTOS RM: 557808 - 2TDSPV

Metamind Solutions

SÃO PAULO

2025

SUMÁRIO

1 Desenho Básico da Arquiteturas

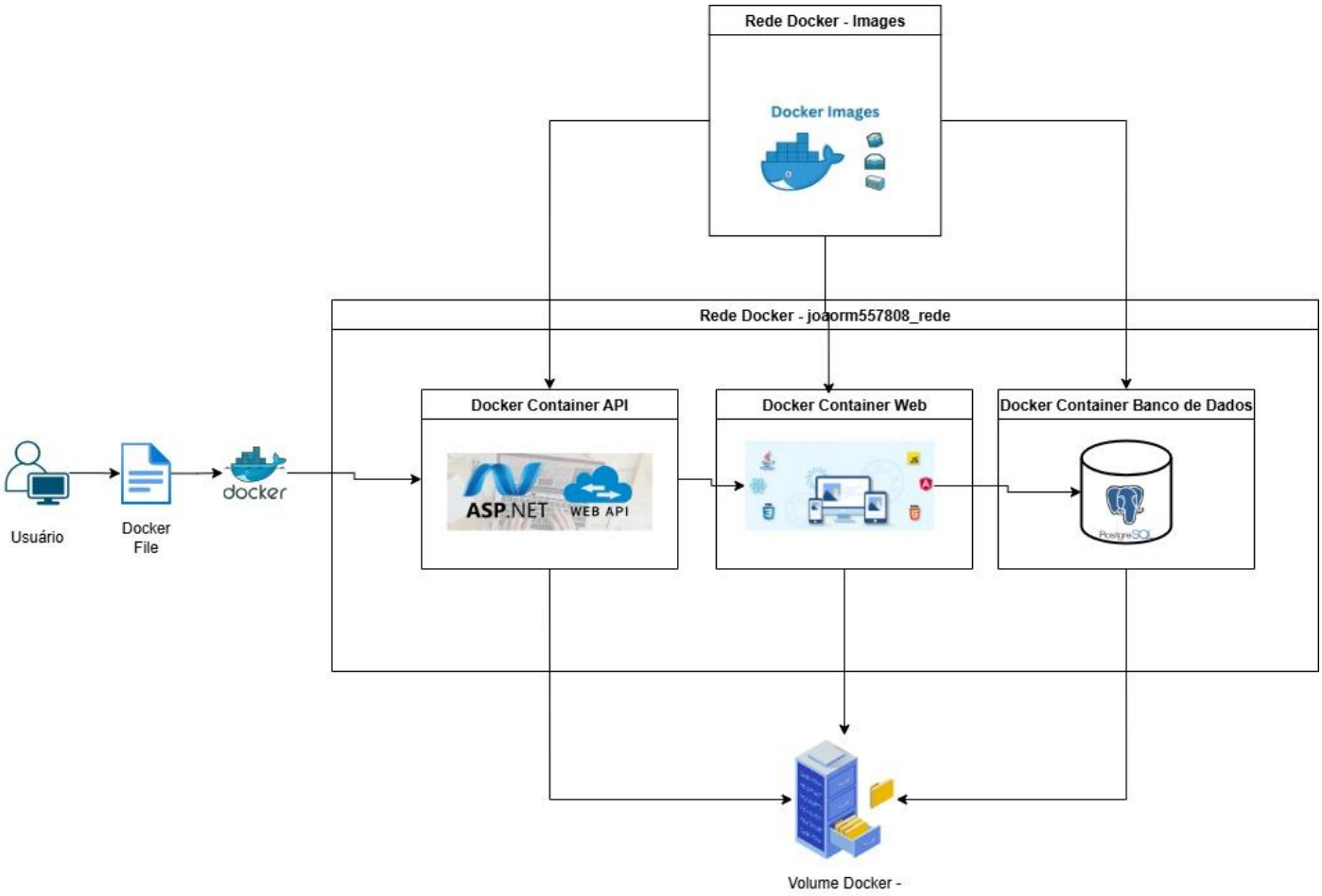

Figura 1: Arquitetura Atual (Docker File)

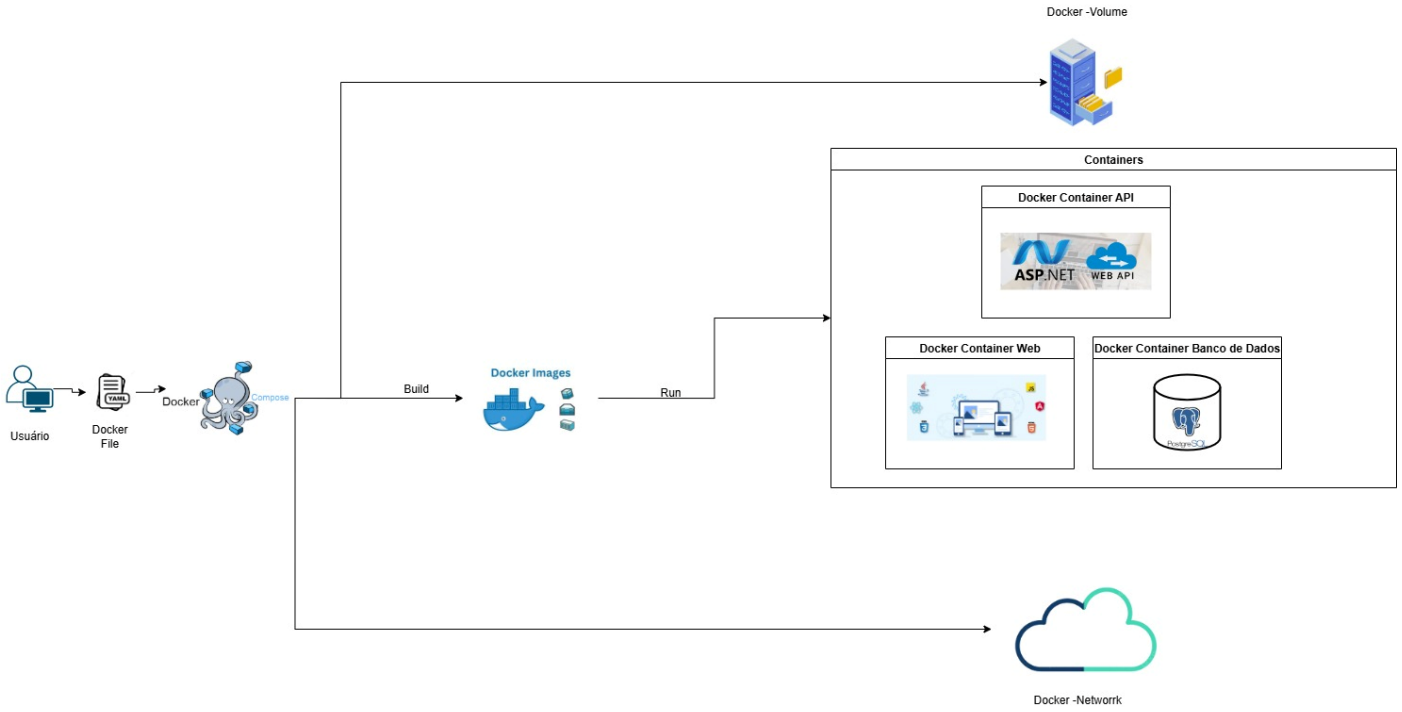

Figura 2: Arquitetura Futura (Docker Compose)

2 Identifique os serviços do projeto

O projeto foi decomposto nos três serviços a seguir, cada um representando um componente lógico da aplicação:

- **db**: Serviço de banco de dados, utilizando a imagem oficial do PostgreSQL. É responsável por toda a persistência e armazenamento dos dados da aplicação.
- **api**: Serviço de backend, contendo a aplicação ASP.NET Core. É responsável por toda a lógica de negócio, regras e pela comunicação com o banco de dados.
- **web**: Serviço de frontend, contendo a aplicação Next.js. É responsável pela interface do usuário e pela interação com o cliente, consumindo os dados expostos pela api.

2.2) Mapeie as dependências entre os serviços

As dependências entre os serviços seguem um fluxo claro e hierárquico, essencial para a correta inicialização e funcionamento da aplicação:

1. O serviço **web (frontend)** depende do serviço **api (backend)**. A interface do usuário precisa que a API esteja em pleno funcionamento para poder buscar e enviar dados.
2. O serviço **api (backend)**, por sua vez, depende do serviço **db (banco de dados)**. A API necessita de uma conexão ativa com o banco de dados para realizar operações de CRUD (Create, Read, Update, Delete).

Essa cadeia de dependências foi implementada no arquivo docker-compose.yml utilizando a diretiva depends_on, garantindo que o banco de dados seja considerado saudável (service_healthy) antes de a API iniciar, e que a API seja considerada saudável antes de o frontend iniciar. A comunicação é habilitada através de uma rede compartilhada (app-net), permitindo que os containers se localizem usando seus nomes de serviço.

2.3) Defina a estratégia de containerização para cada componente

A estratégia de containerização foi definida individualmente para cada serviço, visando otimização, segurança e boas práticas:

- **Serviço db**: Optou-se por utilizar a imagem oficial postgres:16-alpine. A tag 16-alpine foi escolhida por ser uma versão recente, estável e leve (baseada no Alpine Linux), reduzindo a superfície de ataque e o tamanho da imagem. A persistência dos dados é garantida pelo uso de um volume nomeado (db_data), que desvincula os dados do ciclo de vida do container.
- **Serviço api**: Por se tratar de um código customizado (ASP.NET Core), foi criado um Dockerfile específico para ele. Este arquivo é responsável por compilar a aplicação, publicar os artefatos e configurar o ambiente de execução em um container otimizado, garantindo que todas as dependências da aplicação estejam contidas na imagem.
- **Serviço web**: Assim como a API, o frontend é uma aplicação customizada (Next.js) e, portanto, também utiliza um Dockerfile. O processo de build definido no Dockerfile compila a aplicação frontend, gera os arquivos estáticos e prepara o servidor Node.js para servir a aplicação de forma produtiva.

3 Link para vídeo do Projeto e GitHub

CP4 Docker Compose - DIMDIM

ArthurBispo00/-Projeto-DimDim-CP4