Orientações Gerais

Os desafios abaixo devem ser **realizados individualmente** e **disponibilizados em um repositório público no GitHub**.
Cada entrega deve conter um arquivo **`README.md`** com:

- **Descrição da solução**, arquitetura e decisões técnicas.
- **Explicação detalhada do funcionamento** (containers, rede, microsserviços, fluxos, etc).
- **Instruções de execução** passo a passo (como subir os containers e testar).

⚠️ **Atenção:**

- Soluções **idênticas ou excessivamente semelhantes** serão consideradas **plágio**.
- O uso de ferramentas de IA (como ChatGPT, Copilot, etc.) é **permitido apenas como apoio**, não como gerador da solução completa.
  O aluno deve **entender e explicar todo o código** no README.
- O código e explicações serão avaliados com base na **autoria, originalidade e clareza**.

Desafios com Docker (3 desafios)

Desafio 1 — Containers em Rede

Objetivo:
Criar dois containers que se comunicam por uma rede Docker customizada.

Requisitos:

- Um container executa um servidor web (por exemplo, Nginx ou Flask) na porta 8080.
- Outro container realiza requisições HTTP periódicas para o servidor (por exemplo, usando `curl` em loop).
- Criar uma **rede Docker** nomeada e conectar ambos os containers.

- Demonstrar a comunicação e logs da troca de mensagens.

Critérios de avaliação (20 pts):

- [5 pts] Configuração correta da rede Docker.
- [5 pts] Comunicação funcional entre containers.
- [5 pts] Explicação clara no README.
- [5 pts] Organização do projeto e scripts de execução.

Desafio 2 — Volumes e Persistência

Objetivo:
Demonstrar persistência de dados usando volumes Docker.

Requisitos:

- Criar um container com um pequeno banco de dados (por exemplo, SQLite ou PostgreSQL).
- Usar **volumes** para armazenar os dados fora do container.
- Mostrar que, mesmo após remover o container, os dados persistem.
- Opcional: Criar um segundo container para ler os dados persistidos.

Critérios de avaliação (20 pts):

- [5 pts] Uso correto de volumes.
- [5 pts] Persistência comprovada após recriação do container.
- [5 pts] README com explicação e prints/resultados.
- [5 pts] Clareza e organização do código.

Desafio 3 — Docker Compose Orquestrando Serviços

Objetivo:
Usar **Docker Compose** para orquestrar múltiplos serviços dependentes.

Requisitos:

- Criar uma aplicação com **3 serviços** (exemplo: `web`, `db`, `cache`).
- Configurar dependências e variáveis de ambiente via `docker-compose.yml`.
- Testar a comunicação entre os serviços.
- Utilizar `depends_on` e rede interna.

Critérios de avaliação (25 pts):

- [10 pts] Compose funcional e bem estruturado.
- [5 pts] Comunicação entre serviços funcionando.
- [5 pts] README com explicação da arquitetura.
- [5 pts] Clareza e boas práticas.

Desafios com Microsserviços (2 desafios)

Desafio 4 — Microsserviços Independentes

Objetivo:
Criar dois microsserviços independentes que se comunicam via HTTP.

Requisitos:

- Microsserviço A: retorna uma lista de usuários (por exemplo, JSON).
- Microsserviço B: consome o serviço A e exibe as informações combinadas (ex: “Usuário X ativo desde...”).

- Ambos devem ser executáveis via Docker (Dockerfile separado por serviço).
- Comunicação via requisições HTTP (sem gateway ainda).

Critérios de avaliação (20 pts):

- [5 pts] Funcionamento da comunicação entre microsserviços.
- [5 pts] Dockerfiles e isolamento corretos.
- [5 pts] Explicação clara da arquitetura e endpoints.
- [5 pts] Clareza e originalidade da implementação.

Desafio 5 — Microsserviços com API Gateway

Objetivo:
Criar uma arquitetura com **API Gateway** centralizando o acesso a dois microsserviços.

Requisitos:

- Microsserviço 1: fornece dados de usuários.
- Microsserviço 2: fornece pedidos.
- Gateway: expõe endpoints `/users` e `/orders` e orquestra as chamadas aos serviços.
- Todos os serviços rodando em containers via `docker-compose`.

Critérios de avaliação (25 pts):

- [10 pts] Funcionamento do gateway como ponto único de entrada.
- [5 pts] Integração correta entre os serviços.
- [5 pts] README detalhado com explicações e testes.
- [5 pts] Clareza, código organizado e boa documentação.

Avaliação Geral

Critério	Peso	Descrição
Funcionamento técnico	40%	O sistema cumpre os requisitos e roda corretamente em Docker.
Explicação no README	30%	Clareza, domínio do conteúdo e detalhamento da arquitetura.
Organização e boas práticas	20%	Estrutura de pastas, nomes de arquivos, Dockerfiles limpos.
Originalidade / Autoria	10%	Soluções únicas e autoria comprovada.

Entrega

- Enviar o link do repositório GitHub no prazo estabelecido.

O repositório deve conter:

```
/desafio1
/desafio2
/desafio3
/desafio4
/desafio5
README.md
```

- Cada pasta deve conter seus próprios Dockerfiles, `docker-compose.yml` (se houver) e instruções.