Atividade 4 – THREADS, MIDDLEWARE, PARALELISMO, DOCKER (Versão em C++)

Objetivo

Implementar um sistema distribuído todo em **C++ puro**, utilizando **threads** e **containers Docker**, seguindo uma arquitetura **mestre–escravo**.

Funcionalidades

Cliente (Notebook 1):

- Envia um **arquivo** `.txt` que contém letras e números.
- Possui uma **interface gráfica em C++ (Qt/GTK+)** ou versão de linha de comando.
- Apenas envia requisições HTTP (REST) ao **Mestre** e exibe os resultados (não processa os dados localmente).

Servidores (Notebook 2):

Mestre (Container 1)

- Recebe as requisições do Cliente.
- Dispara **duas threads em paralelo**:
  - Cada thread se comunica com um Escravo.
  - Antes de enviar o arquivo, o Mestre consulta se o Escravo está disponível (endpoint `/health`).
- Quando o Mestre receber o resultado dos dois Escravos, ele **combina a resposta** e devolve o resultado consolidado ao Cliente em formato JSON.

Escravos (Containers 2 e 3)

- **Escravo 1:** expõe o endpoint `/letras` que recebe um texto e devolve a **quantidade de letras**.
- **Escravo 2:** expõe o endpoint `/numeros` que recebe um texto e devolve a **quantidade de números**.

Requisitos técnicos

- Linguagem: **C++17 ou superior**.
- Comunicação entre processos: **REST HTTP** (sugestão: usar bibliotecas leves como `cpp-httplib` ou `Boost.Beast`).
- Concorrência: **std::thread**, **std::async** ou equivalente.
- Contêineres: **Docker** com **docker-compose** para orquestrar Mestre + Escravos.
- Cliente pode rodar fora dos containers (Notebook 1) e se comunicar com o Mestre.

Entrega

- Postar no **SIGAA** apenas o link do repositório **GitHub** contendo:
  - Código-fonte do Cliente, Mestre e Escravos.
  - Arquivos **Dockerfile** e **docker-compose.yml**.
  - README explicando o funcionamento, como compilar/executar e exemplos de uso.