Projeto: Middleware para Site de Notícias Distribuído

O projeto visa desenvolver um middleware robusto que gerencie a comunicação entre clientes e servidores de maneira eficiente, tolerante a falhas e escalável. A arquitetura engloba diversas tecnologias e componentes interdependentes para garantir o funcionamento distribuído e otimizado do sistema.

Definição de Tecnologias

- **Backend**: Python com Flask para criação de APIs RESTful.
- **Deploy**:
    - Railway: Deploy contínuo, integração com GitHub e monitoramento.
    - AWS EC2: Hospedagem e escalabilidade.
- **Testes**:
    - **Funcionalidade**: Insomnia e Postman.
    - **Desempenho**: Apache JMeter para carga e stress.
- **Comunicação**: RPC (Remote Procedure Call) para integração eficiente entre os componentes.

Componentes Principais e Funcionalidades

1. Cliente

O cliente interage diretamente com o middleware por meio de um programa Python simples executado em seu terminal.
Funcionalidades:

- **Entrada Inicial**:
    - Coleta nome (validado para não conter números) e idade.
    - Informações mantidas apenas durante a sessão, sem banco de dados.
- **Menu de Categorias**:
    - Exibe opções de notícias (ex.: esportes, política, tecnologia).
    - Determina exibições permitidas com base na faixa etária.
- **Processo de Comunicação**:
    - Envia dados ao proxy para validação e requisição de notícias.
    - Recebe e exibe as notícias formatadas no terminal.
    - Permite nova consulta ou encerramento do programa.

2. Servidor

Cada servidor processa as requisições de clientes e executa regras de negócio.
Funcionalidades:

- **Registro no Sistema**: Envia IP ao Registro de Serviços ao iniciar.
- **Validação de Dados**: Regras como nome sem números e conteúdo restrito por idade.
- **Monitoramento e Sobrecarga**:

    - Conta e reporta threads ativas ao Registro de Serviços.
    - Marca como sobrecarregado quando ultrapassa um limite (ex.: 1000 threads).
- **Comunicação com Proxy**: Processa as requisições e responde com as notícias solicitadas.

3. Registro de Serviços

Centraliza informações sobre servidores ativos e suas condições.
Funcionalidades:

- Recebe e armazena dados de registro (IP e threads ativas).
- Periodicamente verifica a saúde dos servidores.
- Marca servidores como sobrecarregados ou inativos, removendo-os quando necessário.
- Responde às consultas do balanceador de carga com informações atualizadas.

4. Balanceador de Carga

Gerencia a distribuição de requisições entre servidores.
Funcionalidades:

- Seleciona o servidor mais adequado com base na carga e localização.
- Escala novos servidores caso todos estejam sobrecarregados.
- Realiza "downgrade" de servidores extras quando não forem mais necessários.

5. Proxy

Atua como intermediário entre cliente e servidores, mantendo uma fila de mensagens.
Funcionalidades:

- Gerencia threads de clientes conectados.
- Envia requisições do cliente ao balanceador para obter o IP do servidor.
- Encaminha a resposta do servidor ao cliente.
- Garante persistência de conexão durante a sessão, mesmo com falhas.

6. Fila de Mensagem

O proxy guarda a requisição numa fila, e o cliente depois fica perguntando pro proxy se a requisição dele está pronta. Se tiver, o proxy devolve a resposta pro cliente e tira a requisição da fila

7. Monitoramento

- Railway será explorado para replicação e escalabilidade dos componentes (proxy, balanceador e registro de serviços).

Endpoints Principais

Registro de Serviços

Registro do Servidor (POST): `/registrar-servidor`
Exemplo:
```
{
  "ip": "https://servidor-matrix.up.railway.app",
  "threads": 50
}
```

-

Consulta de Servidores (GET): `/consultar-servidores`
Exemplo de Resposta:
```
{
  "https://servidor1.up.railway.app": {"sobrecarga": false, "threads": 20},
  "https://servidor2.up.railway.app": {"sobrecarga": true, "threads": 1001}
}
```

-

Balanceador de Carga

Obtenção de IP do Servidor (GET): `/balancear`
Exemplo de Resposta:
```
{
  "ip": "https://servidor-matrix.up.railway.app"
}
```

-

Proxy

Enfileirar Requisição (POST): `/enqueue_request`
Exemplo:
```
{
  "request_id": "id_unico",
  "method_name": "buscarNoticias",
  "args": ["categoria_tecnologia"]
}
```

-

Servidor

- Listar notícias (GET): `/noticias`
- Consultar categorias (GET): `/categorias`
- Contar threads ativas (GET): `/get_threads`

Testes

1. Funcionais:

- Simular múltiplos clientes e servidores para testar regras de negócio e troca de mensagens.

2. Sobrecarga e Falhas:

- Criar cenários de carga para verificar:
    - Respostas a servidores sobrecarregados.
    - Funcionamento do balanceamento de carga.
    - Redundância e recuperação de falhas.