

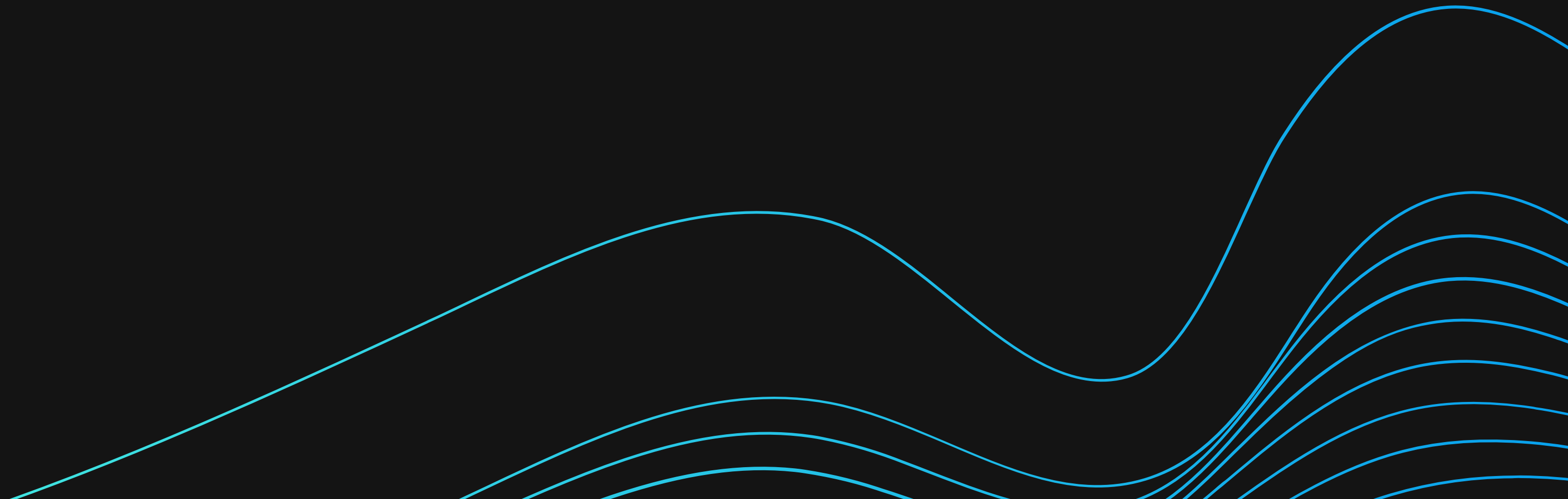
SISTEMAS DISTRIBUÍDOS

An abstract graphic consisting of numerous thin, light blue lines that flow and curve across the dark background, creating a sense of movement and connectivity. The lines originate from the bottom left and fan out towards the top right, with some lines crossing each other.

**GRUPO
MATRIX**

RELEMBRANDO...

Este projeto desenvolve um middleware robusto para um site de notícias distribuído, gerenciando a comunicação entre cliente e servidores de forma eficiente e tolerante a falhas.






O que mudou?


IDEIA INICIAL

- Cliente acessa o site por meio de navegador usando API REST
- Publicação de notícias por usuários com papel de “publicador”
- Banco de dados para armazenar informações dos usuários

AGORA

- 
- Cliente python rodando localmente. A comunicação ocorre via API REST.
 - Notícias estáticas, ou seja, sem publicação de novas notícias
 - Sem utilização de banco de dados

O QUE FIZEMOS?

- **ESPECIFICAÇÃO DAS REGRAS DE NEGÓCIO**
 - **DEFINIÇÃO DE TECNOLOGIAS**
 - Python com o framework Flask para criação de API
 - Railway para deploy contínuo e integração direta com GitHub
 - AWS EC2 para deploy
 - Insomnia e Postman para testes de API
 - Apache JMeter para teste de carga e stress dos serviços
 - **IMPLEMENTAÇÕES**
 - Cliente
 - Servidor
 - Registro de Serviços
 - Parte de proxy/fila de mensagens
 - **CONFIGURAÇÕES E DEPLOY NO RAILWAY e AWS**
 - **TESTES DE API**
- 



CLIENTE

- **Recepção de Dados do Cliente:** O sistema deve receber dados do cliente, como nome e idade, ao iniciar a interação. O nome será validado para garantir que não contenha números, enquanto a idade será usada para determinar as opções de notícias a serem exibidas. Essas informações serão mantidas apenas durante a sessão do cliente, sem persistência entre execuções.
- **Comunicação com o Proxy:** Após a validação dos dados, o cliente enviará ao proxy o tipo de notícia e o proxy deverá retornar as notícias correspondentes ao tipo selecionado. O cliente deverá se comunicar exclusivamente com o proxy para garantir a validação centralizada e o processamento das notícias.
- **Exibição das Notícias:** O cliente deverá exibir as notícias retornadas pelo proxy, formatando-as adequadamente para a visualização no terminal. O cliente poderá então decidir entre finalizar o programa ou selecionar outro tipo de notícia, mantendo o ciclo de interação até que o cliente escolha sair.

CLIENTE

```
PS C:\Users\nicol\Downloads> python3 cliente.py
=====
🚀 DISTRIBUTED NEWS 🚀
=====

Bem-vindo ao DISTRIBUTED NEWS!
Digite seu nome: Nicolas
Digite sua data de nascimento (DD/MM/AAAA): 18/05/2003

Ótimo, Nicolas! Vamos começar.


=====
📰 Escolha o tipo de notícia que você quer ler:
=====
1 - Esportes 🏆
2 - Política 🏛️
3 - Economia 💰
4 - Tecnologia 💻
5 - Entretenimento 🎬
6 - Ciência 🔬
0 - Sair
=====
Digite o número da sua escolha: 4
```



SERVIDOR

- **Registro de Servidor:** Quando o servidor for iniciado, ele deve registrar seu IP no registro de serviços para que os demais serviços possam encontrá-lo e interagir com ele.
- **Gerenciamento de Clientes:** Cada vez que um cliente se conectar ao servidor, uma nova thread será criada para gerenciar a interação e processamento desse cliente.
- **Validação de Regras de Negócio:** O servidor deve validar os dados iniciais do cliente, como, por exemplo, garantir que nomes não contenham números e que o conteúdo seja adequado para a faixa etária do cliente.
- **Monitoramento de Threads:** O servidor deverá contabilizar o número de threads ativas e disponibilizar essa informação para o registro de serviços, que realizará consultas periódicas para monitorar o desempenho e a carga dos servidores.
- **Gerenciamento de Sobrecarga:** O servidor será considerado sobrecarregado quando o número de threads atingir um limite pré-determinado (ex.: 1000 threads). O balanceador de carga será responsável por tomar a decisão de escalar o sistema, acionando a criação de novos servidores quando necessário.
- **Comunicação com Proxy:** O servidor deve se comunicar com o proxy para enviar e receber as validações iniciais dos clientes e o conteúdo das notícias, garantindo que os dados sejam processados corretamente antes de serem entregues ao cliente.

SERVIDOR



```
@app.route('/noticias/categoria/<string:categoria>', methods=['GET'])
def noticias_por_categoria(categoria):
    resultado = buscar_noticias_por_categoria(categoria)
    if isinstance(resultado, str):
        return jsonify({"erro": resultado}), 404
    else:
        return jsonify(resultado), 200
```



```
def registrar_servico():
    try:
        resposta = requests.post("https://registro-de-servicos.up.railway.app/registrar_servidor",
        json={
            "ip": "servidor-matrix.up.railway.app",
            "localizacao": "Brasil",
            "threads": threading.active_count()
        })
        print("Serviço registrado:", resposta.json())
    except Exception as e:
        print(f"Erro ao registrar serviço: {e}")
```



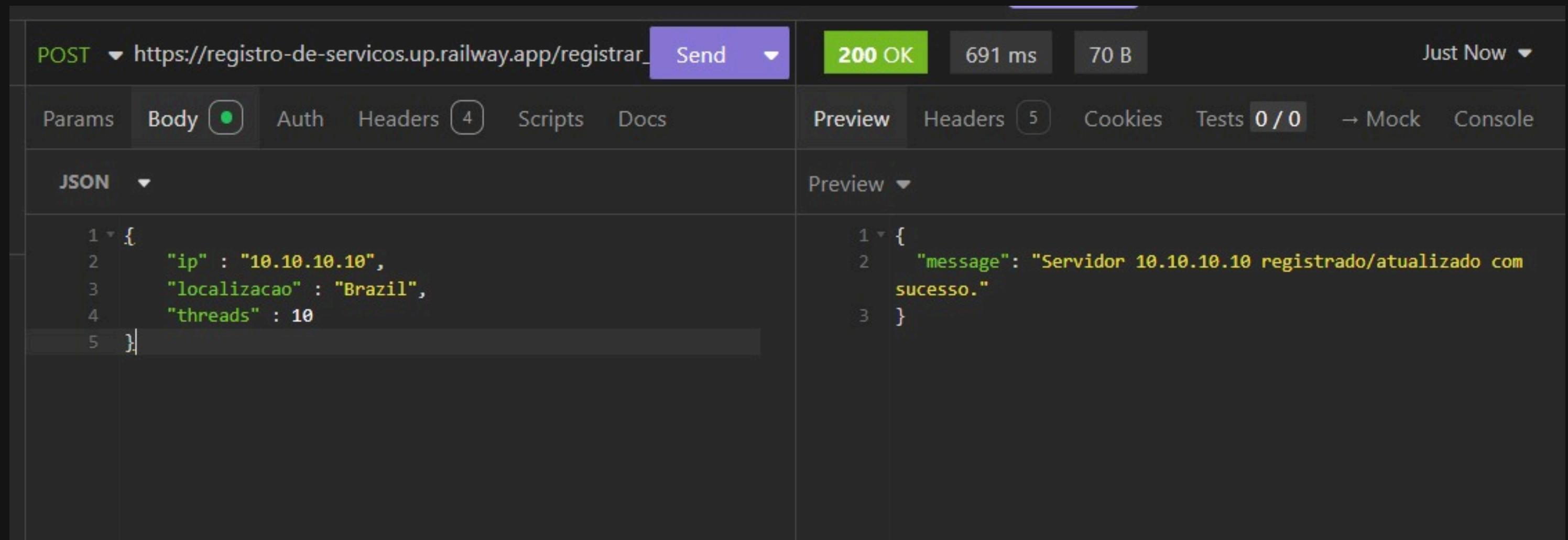

REGISTRO DE SERVIÇOS

- **Recepção de Mensagens do Servidor:** O sistema deve receber mensagens do servidor, contendo sua URL e informações sobre sua localização física. Essas informações devem ser armazenadas em uma estrutura de dado.
- **Monitoramento de Threads do Servidor:** O sistema irá consultar periodicamente o número de threads ativas do servidor. Se o número de threads estiver acima de um limite predefinido, o sistema deverá marcar esse servidor como sobrecarregado utilizando uma flag. Caso o servidor não esteja mais sobrecarregado em consultas subsequentes, a flag deverá ser desmarcada.
- **Gerenciamento de Servidores Inativos:** Caso o sistema tente se comunicar com um servidor e não receba resposta, ele deve remover as informações desse servidor da estrutura de dados utilizada para armazenar os registros.
- **Consulta de Informações pelo Balanceador:** Quando o balanceador de carga precisar de informações sobre um servidor, como sua URL e localização, ele deverá consultar o registro de serviços. O registro de serviços será responsável por fornecer essas informações ao balanceador, garantindo que o balanceador tenha acesso a dados atualizados e precisos sobre a infraestrutura disponível.

REGISTRO DE SERVIÇOS

- URL: registro-de-servicos.up.railway.app
- ENDPOINT para o servidor se registrar (POST): [/registrar-servidor](https://registro-de-servicos.up.railway.app/registrar-servidor)

○



- ENDPOINT para o balanceador consultar servidores (GET): [/consultar-servidores](https://registro-de-servicos.up.railway.app/consultar-servidores)

Sistema-Distribuidos---Projeto ×

Deployments

Variables

Metrics

Settings

 registro-de-servicos.up.railway.app

 US West  1 Replica

ACTIVE



Update registro_de_servicos.py

13 hours ago via GitHub

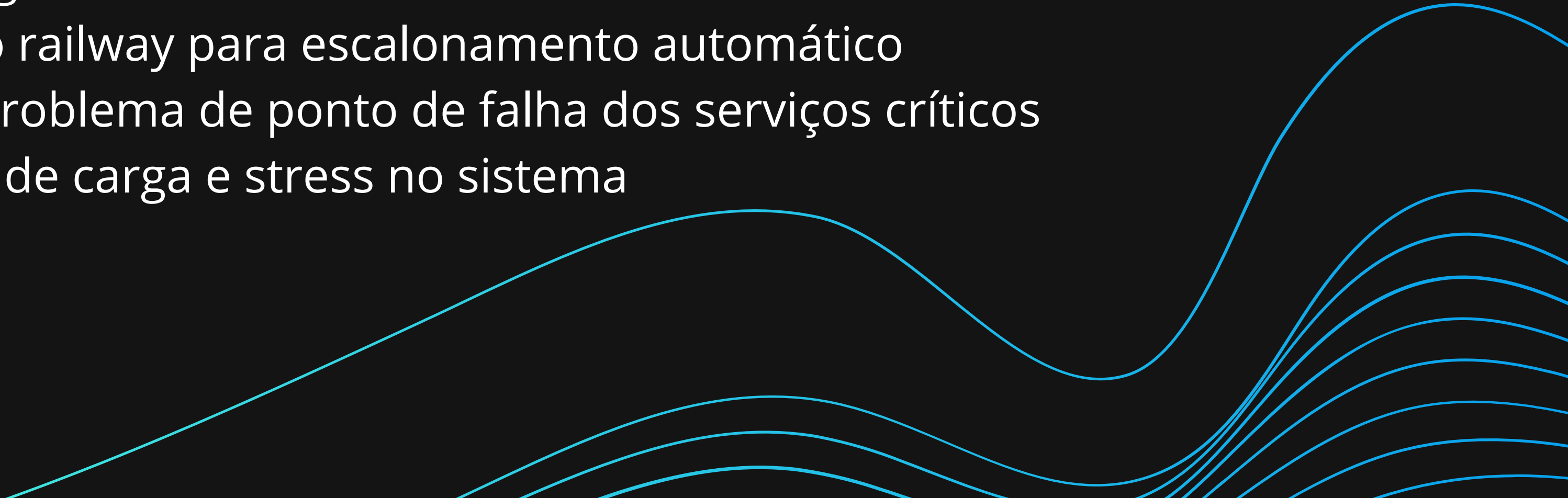
View logs



Deployment successful



O QUE FALTA?

- Implementar
 - Balanceador
 - Terminar o Proxy
 - Terminar Fila de Mensagens
 - Refinar código
 - Configurar o railway para escalonamento automático
 - Resolver o problema de ponto de falha dos serviços críticos
 - Fazer testes de carga e stress no sistema
- 

O QUE PODEMOS MELHORAR?

(se der tempo)

- Acrescentar um banco de dados para persistir os dados dos usuários
 - CRUD para publicar/editar/excluir notícias
 - Filtros para selecionar as notícias (exemplo: por data, por palavra-chave)
 - Sugestões??
- 