Universidade de Brasília

Faculdade de Ciências e Tecnologias em Engenhaira

Laboratório: Virtualização gRPC

PROGRAMAÇÃO PARA SISTEMAS PARALELOS E DISTRIBUÍDOS/T01

ENGENHARIA DE SOFTWARE

Brasília

2025

Universidade de Brasília
Faculdade de Ciências e Tecnologias em Engenhaira

Laboratório: Virtualização gRPC

Artur Vinicius Dias Nunes - 190142421
José Luís Ramos Teixeira - 190057858
Pablo Christianno Silva Guedes - 200042416
Philipe de Sousa Barros - 170154319
Victor de Souza Cabral - 190038900

Professor: Fernando William Cruz

Brasília
2025

Sumário

1 Introdução

Este relatório tem como objetivo descrever as atividades realizadas no laboratório extra-classe proposto na disciplina de Programação para Sistemas Paralelos e Distribuídos (PSPD), oferecida no curso de Engenharia de Software da Universidade de Brasília (UnB). O experimento foi idealizado para permitir aos alunos explorar duas tecnologias fundamentais no desenvolvimento de aplicações modernas: o framework gRPC e as tecnologias de virtualização.

A proposta incluiu a criação de uma aplicação distribuída baseada em microserviços, utilizando o gRPC para comunicação cliente-servidor, e a configuração de uma infraestrutura computacional baseada em virtualização, com o uso de ferramentas como QEMU, KVM e a API Libvirt. A atividade também integrou a construção de uma interface web para disponibilizar o microserviço de dicionário de palavras. No documento é apresentado, em detalhes, a implementação dos componentes solicitados, abordando tanto os aspectos técnicos quanto as dificuldades enfrentadas e uma abre contextualização. O relatório também discute as etapas metodológicas seguidas, as ferramentas utilizadas e comentários sobre a implementação do projeto. Por fim, inclui reflexões dos integrantes do grupo sobre os aprendizados obtidos ao longo da atividade.

Github do projeto: https://github.com/joseluis-rt/gRPC_LabVirtualizacao

Apresentação em Vídeo: https://www.youtube.com/watch?v=s76JdnH8I-g

2 Framework gRPC

2.1 Contextualização

O gRPC foi criado pelo google em 2015 para suprir a necessidade da empresa em conectar microsserviços aos seus datacenters de forma robusta e completa, é um serviço de alto desempenho para atender chamadas RPC (Remote Call Procedures). O RPC é baseado no modelo cliente-servidor, onde o cliente realiza solicitações e o servidor fornece as respostas. Essa comunicação é síncrona, suspendendo o cliente até o retorno do servidor(O que é gRPC?). O conceito de RPC surgiu em 1981, quando Bruce Jay Nelson utilizou o termo para descrever a interação entre dois processos de um mesmo sistema operacional. Hoje, o RPC é amplamente empregado em comunicações de baixo nível. No ecossistema Java, a API JRMI (Java Remote Method Invocation) substituiu o RPC tradicional, permitindo a chamada remota de métodos e classes. JRMI apresenta similaridades com o gRPC, criado pelo Google, mas se concentra em um uso mais orientado a objetos. O gRPC é código aberto se beneficiando desse conceito para facilitar sua evolução, utilização e popularidade, assim tem um suporte interessante para maioria das linguagens de programação mais importantes, oferecendo suporte a load balance, health-check, tracing e autenticação(Introduction to gRPC).

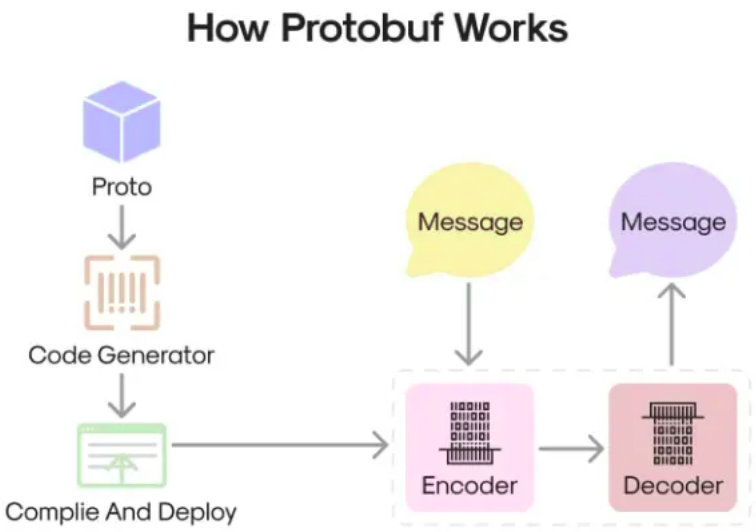
2.2 Elementos constituintes

O gRPC utiliza o HTTP/2 como protocolo de transporte devido às suas características avançadas, permitindo uma interação mais eficiente, escalável e adaptada às exigências de sistemas distribuídos em tempo real. Em contraste, o RPC tradicional, amplamente utilizado no passado, apresenta limitações que tornam o gRPC uma solução mais eficiente e flexível. O protocolo HTTP/2 oferece diversos benefícios que tornam o gRPC uma ferramenta ideal para comunicação em redes modernas. Entre os principais motivos estão: Suporte à Comunicação Bidirecional (Full-Duplex); Compactação de Cabeçalhos: O protocolo utiliza o método HPACK para compactar cabeçalhos de mensagens, reduzindo o overhead de dados. Essa característica complementa a eficiência do Protocol Buffer (Protobuf), utilizado no gRPC para compactar as mensagens de forma binária; Conexões Persistentes; Controle de Fluxo e Prioridade; Compatibilidade e interoperabilidade(2).

O Protocol Buffer (Protobuf) é um formato de serialização de dados estruturados desenvolvido pelo Google. Ele é usado para definir e intercambiar dados de maneira eficiente, compacta e independente de linguagem ou plataforma(2). O Protobuf permite que dados sejam representados em arquivos .proto, onde se define a estrutura das mensagens e suas regras. Esses arquivos são posteriormente compilados em código específico para a linguagem

utilizada (como Java, Python, C++, entre outras), facilitando a integração entre sistemas. Atualmente o grpc utiliza o a versão 3 do protocol buffer que possui syntax mais simples, novas funcionalidades e suporte a várias linguagens(Protobuf vs JSON).

Figura 2.1 – Funcionamento do Protobuf



Fonte: (Protobuf vs JSON)

- **Eficiência e Desempenho:**
  - O Protobuf usa um formato binário que é muito mais compacto e rápido para processar do que formatos como JSON ou XML.
  - Isso é essencial para o gRPC, que visa alta performance em ambientes de baixa latência.
- **Estruturação de Dados:**
  - As mensagens definidas no Protobuf são rigorosamente tipadas, permitindo validação e controle mais robusto sobre os dados transmitidos.
- **Independência de Linguagem e Plataforma:**
  - Como o Protobuf pode ser gerado para diversas linguagens de programação, ele facilita a interoperabilidade em sistemas heterogêneos, uma característica fundamental do gRPC.
- **Definição Simples e Centralizada:**
  - O mesmo arquivo .proto usado para definir as mensagens também especifica os serviços e os métodos expostos pelo gRPC, unificando a configuração do sistema.
- **Extensibilidade:**
  - O Protobuf permite a evolução do esquema sem quebrar compatibilidade, o que é crucial para sistemas distribuídos que podem operar em diferentes versões.

2.3 Comparação com Outras Alternativas de Aplicações Distribuídas

A escolha de um modelo de aplicação distribuída depende de diversos fatores, como desempenho, escalabilidade, compatibilidade e facilidade de implementação. Entre as alternativas mais comuns estão o gRPC, os Web Services baseados em SOAP, as REST APIs e outros modelos de construção de serviços distribuídos. Cada abordagem possui características distintas que a tornam mais ou menos adequada para determinados cenários.

O gRPC e REST, seu principal concorrente, compartilham semelhanças fundamentais em suas arquiteturas de API. Ambos permitem a comunicação entre cliente e servidor com base em regras compartilhadas, independentemente das diferenças internas de cada sistema. Utilizam o protocolo HTTP para troca de dados, embora no gRPC esse processo seja abstraído, enquanto na REST é mais explícito. Ambas as abordagens suportam sistemas distribuídos e escaláveis graças à comunicação assíncrona e ao design sem estado, permitindo o manuseio de um grande volume de solicitações simultâneas e promovendo resiliência. Além disso, são amplamente portáveis, podendo ser implementados em diversas linguagens de programação, o que garante alta interoperabilidade(Qual é a diferença entre gRPC e REST?).

Embora tanto o gRPC quanto a REST ofereçam funcionalidades semelhantes para a construção de APIs, suas arquiteturas diferem significativamente. Na REST, a comunicação segue um modelo unário de solicitação-resposta, limitado a verbos HTTP e orientado a entidades, geralmente usando JSON como formato de dados, que, apesar de legível, é menos eficiente que o Protobuf binário utilizado pelo gRPC. Em contraste, o gRPC suporta múltiplos padrões de comunicação, incluindo streaming bidirecional, orientado a serviços, com geração automatizada de código para cliente e servidor, tornando-o mais rápido e eficiente. Enquanto a REST é frouxamente acoplada, permitindo maior flexibilidade na evolução da API, o gRPC requer um arquivo proto compartilhado, resultando em um acoplamento mais forte. Essas diferenças fazem do gRPC uma escolha ideal para sistemas de alta performance e comunicação avançada, enquanto a REST se destaca pela simplicidade e interoperabilidade(Qual é a diferença entre gRPC e REST?).

Os Web Services baseados em SOAP (Simple Object Access Protocol) utilizam XML como formato de mensagem e frequentemente empregam protocolos como HTTP ou SMTP. Seguindo uma estruturação Rigorosa, as mensagens XML seguem esquemas rígidos definidos em WSDL (Web Services Description Language). Possui padrões Avançados, suportando nativamente a segurança (WS-Security), transações e confiabilidade. Porém conta com alta complexidade, mais difícil de implementar e gerenciar devido à verbosidade do XML e dependência de WSDL. O uso de XML e a complexidade das mensagens tornam o SOAP mais lento, em relação a sua compatibilidade. As solicitações HTTP em APIs tradicionais são enviadas como texto e podem ser lidas e criadas por humanos. Já no gRPC, as mensagens

são codificadas por padrão com Protobuf, que utiliza um formato binário eficiente, mas não legível por humanos(Compare gRPC services with HTTP APIs).

O gRPC enfrenta limitações no suporte direto a navegadores, pois utiliza extensivamente recursos do HTTP/2 que não podem ser controlados pelos navegadores, como exigir HTTP/2 ou acessar frames subjacentes. Para contornar isso no ASP.NET Core, há duas soluções compatíveis com navegadores: o gRPC-Web, que permite que aplicativos web chamem serviços gRPC com Protobuf e um cliente gerado, oferecendo alto desempenho e baixo uso de rede; e o gRPC JSON Transcoding, que transforma serviços gRPC em APIs RESTful com JSON, eliminando a necessidade de um cliente gRPC dedicado. Ambas as soluções têm suporte nativo no .NET, permitindo a integração eficiente de gRPC com aplicações web.

Aspecto	**gRPC**	**REST API**	**Web Services (SOAP)**	**Mensageria (Kafka, etc.)**
Protocolo	HTTP/2	HTTP	HTTP, SMTP	TCP ou protocolos próprios
Formato de Dados	Protobuf (binário)	JSON, XML	XML	Binário ou texto
Latência	Baixa	Moderada	Alta	Variável
Streaming	Suporte completo	Não	Limitado	Sim (assíncrono)
Complexidade	Moderada	Baixa	Alta	Alta
Uso Principal	Sistemas de alta performance	Integrações simples	Sistemas corporativos	Processamento de eventos

Tabela 2.1 – Comparação entre gRPC, REST API, Web Services e Mensageria

Cada tecnologia possui suas vantagens e desvantagens, dependendo do contexto. O gRPC se destaca em cenários de alta performance, comunicação bidirecional e integração entre serviços modernos. Por outro lado, as REST APIs continuam sendo uma escolha popular pela simplicidade e suporte generalizado. Em ambientes corporativos ou legados, SOAP e mensageria ainda desempenham papéis cruciais, especialmente em sistemas que exigem confiabilidade e resiliência.

2.4 Comentários sobre a implementação

O backend foi desenvolvido utilizando Python com o framework Flask, que foi escolhido por ser uma solução simples e eficaz para a criação da API necessária para interagir com o microserviço gRPC. O cliente gRPC foi implementado em Python, enquanto o servidor gRPC foi desenvolvido em C++, o que gerou alguns desafios para a aplicação da framework em duas linguagens diferentes, apesar da simplicidade.

O principal desafio enfrentado pela equipe foi integrar o servidor e o cliente gRPC com o backend. O servidor gRPC, escrito em C++, precisava ser configurado para tratar as requisições do cliente, realizar a lógica de contagem e impressão das palavras, e devolver as respostas adequadas. A comunicação entre o cliente gRPC e o servidor foi o ponto crítico da implementação, pois exigia uma configuração cuidadosa para garantir que as mensagens fossem transmitidas corretamente. O problema mais recorrente do gRPC está relacionado a compilação na linguagem C++ usando o CMAKE, os passos tinham que ser seguidos rigorozamente, mas mesmo assim houve muita dificuldade por partes dos integrantes que

não estiveram tão ativos na parte do desenvolvimento do serviço gRPC. O membro do grupo responsável pela virtualização enfrentou alguns desáfios para configurar uma das máquinas virtuais alguns deles foram: problema de compatibilidade com a framework, build da framework, recursos necessários para suportar o gRPC e pacote cmake antigo o que levou à mudança da versão do ubuntu server para versões mais recentes.

3 Virtualização

3.1 Virtualização com KVM, QEMU e Libvirt

A virtualização é uma tecnologia essencial para a construção de ambientes isolados e controlados, permitindo a execução de múltiplas instâncias de sistemas operacionais em uma única máquina física. No contexto deste trabalho, utilizamos as ferramentas KVM, QEMU e a API Libvirt para configurar e gerenciar as máquinas virtuais (VMs) necessárias para a execução da aplicação distribuída.

3.1.1 Arquitetura Interna do KVM/QEMU

O KVM (Kernel-based Virtual Machine) é uma solução de virtualização baseada no Linux que permite a execução de múltiplas máquinas virtuais em um único host físico, utilizando o processador de hardware para criar máquinas virtuais isoladas, com alto desempenho e eficiência. O QEMU (Quick Emulator) é um emulador que, junto com o KVM, oferece a capacidade de executar virtualizações de sistemas operacionais completos.

Essas ferramentas são gerenciadas pelo Libvirt, que oferece uma API para facilitar a administração e automação de tarefas relacionadas à criação, configuração e monitoramento das VMs.

3.1.2 Uso de Firmware em Máquinas Virtuais

Durante a configuração das VMs, foi necessário configurar o firmware para garantir o funcionamento correto. Utilizamos o **Coreboot** e o **SeaBIOS** para inicializar as máquinas virtuais, sendo o Coreboot mais leve e rápido, enquanto o SeaBIOS oferece compatibilidade com BIOS tradicional.

3.1.3 Comandos Usados no Processo

O processo de criação e configuração das máquinas virtuais foi realizado principalmente pelo **virt-manager**, que fornece uma interface gráfica para facilitar o gerenciamento. No entanto, também usamos o **virsh** para comandos rápidos e para verificar o status da rede, entre outras tarefas.

3.1.3.1 1. Configuração das Redes Virtuais (LAN1 e LAN2)

Primeiro, configuramos duas redes virtuais: **LAN1** (física, conectando as VMs à rede Wi-Fi do host) e **LAN2** (virtual, conectando as VMs entre si). A configuração das redes foi

feita no **virt-manager**, com os seguintes passos:

- Criamos as redes usando o **virt-manager**:
  - **LAN1** foi configurada para se conectar à rede Wi-Fi do host físico (`wlp2s0`).
  - **LAN2** foi configurada com o endereço IP estático `192.168.200.1/24`, funcionando como um gateway para as VMs.
- A partir do **virt-manager**, configuramos os NICs das VMs para cada rede:
  - **VM01** foi configurada para ter duas interfaces de rede: uma para a **LAN1** (com o IP `192.168.0.130/24`) e outra para a **LAN2** (com o IP estático `192.168.200.10/24`).
  - **VM02** foi configurada para usar apenas a **LAN2**, com o IP estático `192.168.200.20/24`.
- Ajustamos a configuração de rede estática nas VMs usando o `netplan`, acessando os arquivos `/etc/netplan/*.yaml` de cada VM e configurando as interfaces de rede. Após as edições, executamos `sudo netplan apply` para aplicar as configurações.

3.1.3.2 2. Comandos no Terminal (virsh)

Para verificar o estado da rede e das VMs, usamos os seguintes comandos **virsh**:

- Para verificar as redes virtuais ativas:

```
sudo virsh net-list --all
```

- Para verificar as máquinas virtuais ativas:

```
sudo virsh list --all
```

- Para iniciar ou reiniciar uma VM:

```
sudo virsh start VM01
sudo virsh start VM02
```

Esses comandos foram úteis para monitorar e garantir que as redes virtuais e as VMs estivessem funcionando corretamente.

3.1.3.3 3. Verificação da Rede

Para garantir que as VMs estavam se comunicando corretamente pela **LAN2**, realizamos testes de ping entre as máquinas:

- **VM01** (lan2): Pingou **VM02** (lan2) com sucesso.
- **VM02** (lan2): Pingou **VM01** (lan2) com sucesso.
- **VM01** (lan1): Pingou a rede Wi-Fi do host físico sem problemas.

Esse processo foi crucial para garantir a conectividade entre as VMs e o host físico, bem como entre as próprias VMs.

3.1.4 Dificuldades Encontradas

Durante a configuração das redes e VMs, encontramos alguns desafios, principalmente relacionados à configuração correta das interfaces de rede e ao ajuste da configuração de IP estático. A principal dificuldade foi garantir que ambas as VMs tivessem acesso à **LAN2** e que a **VM01**, com duas interfaces de rede, estivesse corretamente configurada para rotear os pacotes entre as redes. O uso do **virsh** foi essencial para verificar se as redes estavam funcionando corretamente e para resolver problemas de configuração.

Além disso, o processo de instalação do Ubuntu nas VMs foi desafiador em termos de tempo, especialmente ao tentar usar versões diferentes (Ubuntu 20.04 e 22.04) para testar compatibilidade com o **gRPC C++**.

3.1.5 Resultados Alcançados

Após a configuração das VMs e das redes virtuais, conseguimos implementar com sucesso a comunicação entre as VMs usando o gRPC. A **VM01** (cliente) se comunica corretamente com a **VM02** (servidor) através da **LAN2**. A configuração das redes no **virt-manager** e as edições no `netplan` garantiram que as VMs tivessem conectividade estável e eficiente. O uso do **virsh** para monitorar e gerenciar as VMs foi fundamental para garantir que o ambiente estivesse funcional.

4 Conclusão

iniciar com um texto conclusivo sobre o experimento e subseções para que cada aluno possa manifestar sua opinião e aprendizados específicos sobre o que foi feito, além de uma nota de auto-avaliação, em função do grau de envolvimento com o trabalho.

O experimento desenvolvido no contexto da disciplina de Programação para Sistemas Paralelos e Distribuídos (PSPD) permitiu uma imersão prática nas tecnologias de gRPC e virtualização, proporcionando aos participantes uma compreensão mais aprofundada sobre a implementação de aplicações distribuídas baseadas em microserviços. A utilização do gRPC demonstrou sua eficiência na comunicação entre clientes e servidores, evidenciando suas vantagens em termos de desempenho e simplicidade em comparação com outras abordagens tradicionais. A configuração de ambientes virtualizados, utilizando ferramentas como QEMU, KVM e Libvirt, reforçou a importância da infraestrutura virtualizada para a execução e escalabilidade de aplicações distribuídas.

Durante o desenvolvimento do experimento, os integrantes do grupo enfrentaram desafios técnicos relacionados à configuração dos ambientes e à integração dos componentes, o que contribuiu significativamente para o aprendizado prático. O relatório documenta detalhadamente as etapas do projeto, fornecendo uma visão abrangente sobre os aspectos técnicos e metodológicos adotados. A atividade também destacou a importância do trabalho colaborativo e da documentação estruturada para o sucesso do projeto.

4.0.1 Opinião, aprendizados individuais e autoavaliação

Nesta subseção, os membros do grupo compartilham suas reflexões sobre a experiência adquirida e os conhecimentos assimilados ao longo do experimento.

Artur: Foi bem interessante pode ter um contato com os serviços gRPC e de virtualização, e poder entender como utiliza-los em um determinado contexto. Esse trabalho foi bem interessante porém muiuto desafiador já que enfretamos vários problemas de configuração ao longo do processo, enquanto tentavamos entender melhor como essas tecnologias funcionavam e se comunicavam. O mais importante foi poder absorver conhecimento da interação e integração das partes, com a equipe, já que não tinha como todos participarem ativamente de todas as partes do trabalho. Autoavaliação: Boa.

José Luís: O trabalho foi um desafio, principalmente pela parte de configuração das VMs e integração do gRPC. Enfrentei diversas dificuldades relacionadas ao gRPC C++, que demandaram muito tempo, já que os erros só eram detectados após longos períodos de compilação, o que levou a alguns atrasos no progresso. A configuração de rede entre as VMs foi outro ponto que exigiu atenção, mas com paciência consegui resolver. O aprendizado foi

significativo, especialmente ao trabalhar com a configuração do ambiente no Ubuntu 22.04 para rodar o gRPC, que antes não funcionava na versão 20.04. Embora o trabalho tenha sido mais complexo no lado do servidor, a parte do cliente em Python foi mais simples e fluída. Fiquei bastante satisfeito com a conclusão do trabalho e o funcionamento do sistema, especialmente considerando os obstáculos enfrentados. No entanto, acredito que o projeto poderia estar um pouco mais polido se tivéssemos mais tempo, mas isso faz parte do processo. Ainda assim, aprendi muito com o uso do gRPC e com a resolução dos desafios que surgiram ao longo do caminho. Autoavaliação: Excelente.

Pablo: O trabalho propôs um desafio interessante que permitiu muita aprendizagem, enfretamos muitas dificuldades em relação a erros, o que demandou bastante tempo do desenvolvimento levando a atrasos. Tendo isso em vista, apesar das dificuldades, o grupo teve uma ótima comunicação e foi muito satisfatório ver o sistema em funcionamento. Os maiores aprendizados que obtive foi em relação a implementação dos serviços gRPC no qual atuei ativamente, principalmente em relação a build em C++ que não estava familiarizado, também pude aprender bastante sobre virtualização apesar de não ter atuado com tanto afinco nessa parte, prestei assistência na resolução dos problemas encontrados. Autoavaliação: Excelente.

Philipe: Minha participação no projeto concentrou-se na documentação das atividades realizadas, na confecção deste relatório e registro detalhado de cada etapa do experimento. A elaboração do relatório permitiu um entendimento teórico mais aprofundado das tecnologias abordadas, mesmo sem uma participação direta na implementação prática, no entanto, reconheço que uma participação mais ativa no desenvolvimento teria agregado maior valor ao meu aprendizado prático. Autoavaliação: Regular.

Victor: Tive contato com tecnologias que não tenho tanta familiaridade e pude aprender bastante no processo de desenvolvimento do trabalho. Atuei no desenvolvimento do frontend e integração do próprio com o backend e esse último com o cliente e servidor, além de ajustar a correta exibição dos resultados na interface. Garantir que todas as partes estivessem configuradas e interagindo corretamente foi um desafio. Pude aprender muito também da configuração do serviços gRPC e virtualização através de interações com os outros membros. Autoavaliação: Boa.

Referências

COMPARE gRPC services with HTTP APIs. [*s.d.*]. Acesso em: 17 jan. 2025. Disponível em: https://learn.microsoft.com/en-us/aspnet/core/grpc/comparison?view=aspnetcore-9.0. Citado na p. 7.

CORE concepts, architecture and lifecycle. [*s.d.*]. Acesso em: 15 jan. 2025. Disponível em: https://grpc.io/docs/what-is-grpc/core-concepts/. Citado na p. 4.

INTRODUCTION to gRPC. [*s.d.*]. Acesso em: 15 jan. 2025. Disponível em: https://grpc.io/docs/what-is-grpc/introduction/. Citado na p. 4.

O que é gRPC? [*s.d.*]. Acesso em: 15 jan. 2025. Disponível em: https://coodesh.com/blog/dicionario/o-que-e-grpc/. Citado na p. 4.

PROTOBUF vs JSON. [*s.d.*]. Acesso em: 16 jan. 2025. Disponível em: https://protobuf.dev/overview/. Citado na p. 5.

PROTOBUF vs JSON. [*s.d.*]. Acesso em: 16 jan. 2025. Disponível em: https://lab.wallarm.com/what/protobuf-vs-json/#:~:text=If%20your%20project's%20foundation%20lies,stands%20as%20the%20favourable%20alternative. Citado na p. 5.

QUAL é a diferença entre gRPC e REST? [*s.d.*]. Acesso em: 17 jan. 2025. Disponível em: https://aws.amazon.com/pt/compare/the-difference-between-grpc-and-rest/. Citado na p. 6.


UnB