1. **Introdução**

Um repositório de dados é um local onde as informações são armazenadas. Essa infraestrutura tem a função de coletar, armazenar e gerenciar dados para preservação e compartilhamento futuro.

O objetivo de um repositório de dados é aceitar dados externos para uso de uma empresa, instituição ou indivíduo. Isso significa que os dados no repositório são padronizados e organizados de maneira que possam ser facilmente pesquisados e utilizados por outros e para o efeito de padronização são usados alguns modelos conhecidos por modelos de acesso remoto a dados que serão abordados mais a seguir no desenvolvimento do trabalho. Repositórios remotos guiam uma grande quantidade de sites e aplicativos web que usam dos dados disponíveis nesses repositórios com base no uso de APIs, por exemplo. Eles desempenham um papel fundamental ao fornecer um local centralizado para armazenar e compartilhar dados, facilitando a colaboração e a reutilização de informações em diferentes contextos.

Este trabalho de pesquisa tem como objectivo abordar aspectos que valorizam a utilização destes repositórios, apontar vantagens e conhecer alguns detalhes que caracterizam estes repositórios.

1. **Modelos de acesso remoto a repositórios de dados** 
   1. **SOAP (Simple Object Access Protocol)**

É um protocolo de mensagem que permite que os elementos distribuídos de um aplicativo se comuniquem. O SOAP pode ser transportado por uma variedade de [protocolos](https://www.techtarget.com/searchnetworking/definition/protocol) padrão, incluindo o [HTTP](https://www.techtarget.com/whatis/definition/HTTP-Hypertext-Transfer-Protocol) (Hypertext Transfer Protocol) relacionado à Web.

O SOAP foi desenvolvido como uma linguagem intermediária para aplicações que possuem diferentes linguagens de programação, permitindo que essas aplicações se comuniquem entre si pela internet. O SOAP é flexível e independente, o que permite que os desenvolvedores escrevam interfaces de programação de aplicativos (APIs) SOAP em diferentes linguagens, além de adicionar recursos e [funcionalidades](https://www.techtarget.com/searchapparchitecture/definition/application-program-interface-API).

SOAP é um protocolo leve usado para criar APIs da Web, geralmente com [XML](https://www.techtarget.com/whatis/definition/XML-Extensible-Markup-Language) (Extensible Markup Language). Ele suporta uma ampla gama de protocolos de comunicação através da Internet, HTTP, SMTP (Simple Mail Transfer Protocol) e [Protocolo de Controle de Transmissão](https://www.techtarget.com/searchnetworking/definition/TCP). A abordagem SOAP define como uma mensagem SOAP é processada, os recursos e módulos incluídos, os protocolos de comunicação suportados e a construção de mensagens SOAP. O SOAP usa o Conjunto de Informações XML como um formato de mensagem e depende de protocolos de camada de aplicativo, como HTTP, para transmissão e negociação de mensagens.

* + 1. **Características**

O SOAP fornece a camada de Protocolo de Mensagens de uma [pilha de protocolos de serviços Web](https://en.wikipedia.org/wiki/Web_services_protocol_stack) para serviços Web. É um protocolo baseado em XML que consiste em três partes:

* um envelope, que define a estrutura da mensagem[[1]](https://en.wikipedia.org/wiki/SOAP#cite_note-1) e como processá-la
* um conjunto de regras de codificação para expressar instâncias de tipos de dados definidos pelo aplicativo
* uma convenção para representar os convites e respostas aos procedimentos

O SOAP tem três **características principais**:

* **extensibilidade** (segurança e [WS-Addressing](https://en.wikipedia.org/wiki/WS-Addressing) estão entre as extensões em desenvolvimento)
* **neutralidade** (SOAP pode operar sobre qualquer protocolo, como [HTTP,](https://en.wikipedia.org/wiki/HTTP) [SMTP,](https://en.wikipedia.org/wiki/SMTP) [TCP,](https://en.wikipedia.org/wiki/Transmission_Control_Protocol) [UDP](https://en.wikipedia.org/wiki/SOAP-over-UDP))
* **independênci**a (SOAP permite qualquer [modelo de programação](https://en.wikipedia.org/wiki/Programming_model))
  + 1. **Funcionamento**

As solicitações SOAP são fáceis de gerar e processar respostas. Primeiro, uma solicitação para um serviço é gerada por um [cliente](https://www.techtarget.com/searchenterprisedesktop/definition/client) usando um documento XML. Em seguida, um cliente SOAP envia o documento XML para um servidor SOAP. Quando o servidor recebe a mensagem SOAP, ele envia a mensagem como uma chamada de serviço para o aplicativo do lado do servidor solicitado. Uma resposta contendo os parâmetros solicitados, valores de retorno e dados para o cliente é retornada primeiro para o manipulador de solicitação SOAP e, em seguida, para o cliente solicitante. As solicitações e respostas SOAP são transportadas usando [HTTPS](https://www.techtarget.com/searchsoftwarequality/definition/HTTPS) (Hypertext Transfer Protocol Secure) ou um protocolo semelhante como HTTP.

* + 1. **Vantagens e desvantagens do SOAP**

O SOAP é parte integrante da arquitetura orientada a serviços ([SOA)](https://www.techtarget.com/searchapparchitecture/definition/service-oriented-architecture-SOA) e das especificações de [serviços da Web](https://www.techtarget.com/searchapparchitecture/definition/Web-services). As vantagens do SOAP incluem o seguinte:

* Independente de plataforma e sistema operacional. O SOAP pode ser transportado através de uma variedade de protocolos, permitindo a comunicação entre aplicativos com diferentes linguagens de programação no Windows e no Linux.
* Funciona no protocolo HTTP. Embora o SOAP funcione com muitos protocolos diferentes, HTTP é o protocolo padrão usado por aplicativos Web.
* Pode ser transmitido através de diferentes dispositivos de rede e segurança. O SOAP pode ser facilmente passado através [de firewalls](https://www.techtarget.com/searchsecurity/definition/firewall), onde outros protocolos podem exigir uma acomodação especial.
  + 1. **As desvantagens, no entanto, incluem o seguinte**:
* Nenhuma disposição para a transmissão de dados por referência. Isso pode causar problemas de sincronização se várias cópias do mesmo objeto forem passadas simultaneamente.
* Velocidade. A estrutura de dados do SOAP é baseada em XML. O XML é em grande parte legível por humanos, o que torna bastante fácil entender uma mensagem SOAP. No entanto, isso também torna as mensagens relativamente grandes em comparação com o Common Object Request Broker Architecture (CORBA) e seu protocolo [RPC (chamada](https://www.techtarget.com/searchapparchitecture/definition/Remote-Procedure-Call-RPC) de procedimento remoto) que acomodará dados binários. Por isso, CORBA e RPC são mais rápidos.
* Não tão flexível quanto outros métodos. Embora o SOAP seja flexível, métodos mais recentes, como a arquitetura RESTful, usam XML, [JavaScript Object Notation](https://www.theserverside.com/definition/JSON-Javascript-Object-Notation), [YAML](https://www.techtarget.com/searchitoperations/definition/YAML-YAML-Aint-Markup-Language) ou qualquer analisador necessário, o que os torna mais flexíveis do que o SOAP.
  1. **GraphQL**

GraphQL é uma linguagem de [manipulação](https://en.wikipedia.org/wiki/Query_language) e [consulta](https://en.wikipedia.org/wiki/Data_manipulation_language) [de dados de código aberto](https://en.wikipedia.org/wiki/Open-source_software) para [APIs](https://en.wikipedia.org/wiki/API) e um [mecanismo de tempo de execução](https://en.wikipedia.org/wiki/Runtime_system) de consulta.

O GraphQL permite a busca de dados [declarativos](https://en.wikipedia.org/wiki/Declarative_programming) onde um [cliente](https://en.wikipedia.org/wiki/Client%E2%80%93server_model) pode especificar exatamente quais dados ele precisa de uma API. Em vez de vários [pontos de extremidade](https://en.wikipedia.org/wiki/Service-oriented_architecture) que retornam dados separados, um servidor GraphQL expõe um único ponto de extremidade e responde com precisão com os dados solicitados por um cliente. [[2]](https://en.wikipedia.org/wiki/GraphQL#cite_note-2) Como um servidor GraphQL pode buscar de fontes de dados separadas e apresentar os dados em um [gráfico](https://en.wikipedia.org/wiki/Graph_(abstract_data_type)) unificado, ele não está vinculado a nenhum [banco de dados](https://en.wikipedia.org/wiki/Database) ou mecanismo de armazenamento específico.

O GraphQL suporta leitura, gravação (mutação) e assinatura de alterações em dados (atualizações em tempo real – comumente implementadas usando [WebSockets](https://en.wikipedia.org/wiki/WebSocket)). [[8]](https://en.wikipedia.org/wiki/GraphQL#cite_note-8) Um serviço GraphQL é criado definindo tipos com campos e, em seguida, fornecendo funções para resolver os dados de cada campo. Os tipos e campos compõem o que é conhecido como a definição de esquema. As funções que recuperam e mapeiam os dados são chamadas de resolvedores.

Depois de ser validada em relação ao esquema, uma consulta GraphQL é executada pelo servidor. O servidor retorna um resultado que espelha a forma da consulta original, normalmente como [JSON](https://en.wikipedia.org/wiki/JSON).

* + 1. **Vantagem do GraphQL**
* Busca eficiente de dados - no GraphQL, os clientes têm a capacidade de especificar exatamente os dados de que precisam, evitando assim os problemas comuns de busca excessiva e sub-busca que geralmente surgem com [APIs RESTful](https://www.geeksforgeeks.org/rest-api-introduction/). Os clientes podem pedir apenas as informações de que necessitam. Isso interrompe problemas de muitos dados (busca excessiva) ou dados insuficientes (busca insuficiente) que geralmente acontecem com APIs RESTful. Os clientes podem solicitar apenas os dados de que precisam, eliminando problemas de busca excessiva e sub-busca comuns em APIs RESTful.
* Esquema fortemente tipado **-** Usar APIs GraphQL tem muitas vantagens para os desenvolvedores. O esquema fortemente tipado atua como um plano claro, delineando a estrutura precisa e os tipos de dados para clientes e servidores. Isso ajuda a eliminar a confusão e garante uma comunicação suave entre diferentes partes do aplicativo. Ao contrário das APIs REST tradicionais, que podem ser ambíguas.
* Agrupamento de múltiplas fontes de dados - O GraphQL é uma tecnologia que permite reunir informações de diferentes lugares em um local fácil de usar.Ele nos permite obter todos os dados que precisamos sem ter que ir a vários lugares diferentes. O GraphQL simplifica a obtenção de dados.
* Versionamento e Evolução - as estruturas do GraphQL podem mudar ao longo do tempo sem impedir que os clientes atuais trabalhem. Isso acontece porque o GraphQL permite verificar quais campos estão disponíveis. Os desenvolvedores também podem remover campos lentamente em vez de removê-los todos de uma só vez.
* Número reduzido de endpoints - quando se trata de GraphQL, um dos principais benefícios é sua simplicidade no gerenciamento de APIs. Isso ocorre porque o GraphQL expõe um único ponto de extremidade, o que contrasta com as APIs RESTful que podem ter vários pontos de extremidade.
  + 1. **Desvantagens do GraphQL**
* Sobrecarga de complexidade - criar um servidor GraphQL e configurar o esquema é mais complexo e envolvido do que criar APIs RESTful. Para os desenvolvedores, é necessário se familiarizar com os conceitos do GraphQL e seguir as práticas recomendadas. No entanto, uma vez que os fundamentos são compreendidos, o GraphQL oferece recursos poderosos que simplificam a busca de dados e permitem consultas eficientes. A configuração adicional e a complexidade inicial são contrabalançadas pela flexibilidade e pelos ganhos de desempenho. Aprender GraphQL pode ser complicado.
* Potencial excesso de busca - pedir muitas informações ao mesmo tempo pode afetar negativamente o desempenho, mesmo com o GraphQL. É preciso cuidado extra para evitar isso. Se deve certificar de otimizar as consultas e planejar o esquema cuidadosamente.
* Preocupações de segurança - as APIs do GraphQL podem estar abertas a riscos de segurança. Esses riscos incluem ataques excessivos de profundidade ou complexidade de consulta. Os desenvolvedores devem implementar medidas de segurança para evitar vulnerabilidades. A limitação de taxa e a validação de consultas são duas medidas de segurança importantes. A limitação de taxa controla quantas solicitações podem ser feitas em um determinado período de tempo. Isso evita sobrecarregar o sistema com muitas solicitações. A validação de consulta verifica as consultas para garantir que elas sigam regras e limites.
* Sobrecarga de design de esquema - ter um esquema GraphQL bem construído é realmente importante. Se o esquema não for projetado corretamente, ele pode causar grandes problemas, como desempenho lento e não ser capaz de lidar com muito tráfego e usuários.
* Curva de aprendizado - aprender GraphQL pode ser um desafio para desenvolvedores e equipes. Envolve dominar novas ideias e técnicas, o que pode ser difícil, especialmente se eles já são habilidosos com APIs RESTful.
  1. **gRPC**

gRPC (gRPC Remote Procedure Calls[[2]](https://en.wikipedia.org/wiki/GRPC#cite_note-2)) é uma estrutura de [RPC (Remote Procedure Call](https://en.wikipedia.org/wiki/Remote_procedure_call)) de alto desempenho [de código aberto multiplataforma](https://en.wikipedia.org/wiki/Cross-platform). O gRPC foi inicialmente criado pelo [Google](https://en.wikipedia.org/wiki/Google), que usou uma única infraestrutura RPC de uso geral chamada Stubby para conectar o grande número de [microsserviços](https://en.wikipedia.org/wiki/Microservices) executados dentro e através de seus [data centers](https://en.wikipedia.org/wiki/Data_centers) a partir de cerca de 2001

 O resultado foi o gRPC, que agora é usado em muitas organizações, além do Google, para potencializar casos de uso de microsserviços até a "última milha" da computação (celular, web e Internet das Coisas). Ele usa [HTTP/2](https://en.wikipedia.org/wiki/HTTP/2) para transporte, [buffers de protocolo](https://en.wikipedia.org/wiki/Protocol_Buffers) como a [linguagem de descrição da interface](https://en.wikipedia.org/wiki/Interface_description_language) e fornece recursos como autenticação, streaming bidirecional e [controle de fluxo](https://en.wikipedia.org/wiki/Flow_control_(data)), ligações de bloqueio ou não bloqueio e cancelamento e tempos limites. Ele gera ligações de cliente e servidor de plataforma cruzada para vários idiomas. Os cenários de uso mais comuns incluem a conexão de serviços em uma arquitetura de estilo de microsserviços ou a conexão de clientes de dispositivos móveis a serviços de back-end. [[4]](https://en.wikipedia.org/wiki/GRPC#cite_note-:0-4)

A ideia base do gRPC era ser muito mais performático do que a sua contraparte REST por ser baseado no HTTP/2. Este conjunto de ferramentas torna possível que o gRPC seja utilizado em diversas linguagens ao mesmo tempo com um overhead muito baixo enquanto continua sendo mais rápido e mais eficiente do que as demais arquiteturas de chamadas de rede.

* + 1. **Vantagens do gRPC**

Como pudemos ver, o gRPC tem várias vantagens sobre o modelo ReST tradicional:

* Mais leve e mais rápido por utilizar codificação binária e HTTP/2
* Multi plataforma com a mesma interface de contratos
* Funciona em muitas plataformas com pouco ou nenhum overhead
* O código é auto documentado
* Implementação relativamente fácil depois do desenvolvimento inicial
* Excelente para trabalhos entre times que não vão se encontrar, principalmente para definir contratos de projetos open source.
  + 1. **Desvantagens**

Assim como toda a tecnologia, o gRPC não é uma bala de prata e não resolve todos os problemas, temos alguns defeitos:

* O protobuf não possui um package manager para poder gerenciar as dependências entre arquivos de interface
* Exige uma pequena mudança de paradigma em relação ao modelo ReST
* Curva de aprendizado inicial é mais complexa
* Não é uma especificação conhecida por muitos
* Por conta de não ser muito conhecido, a documentação é esparsa
* A arquitetura de um sistema usando gRPC pode se tornar um pouco mais complexa
  1. **REST**

O REST é uma forma de desenvolvimento de criar sistemas distribuídos que funcionam em conjunto através da internet usando links. É um modelo que implementa regras e formatos comuns aos desenvolvedores, facilitando o desenvolvimento de novos softwares por via destes.

Um serviço REST pode ser implementado usando o Microsoft ASP .NET Core, por exemplo, e podem ser consumidos usando qualquer linguagem de programação, se utilizando ainda de ferramentas presentes nas linguagens que permitem gerar solicitações HTTP e analisar respostas que são retornadas por via deste mesmo protocolo. Essas requisições HTTP sao enviadas na forma de URIs que identificam os recursos, juntamente com o tipo de requisição HTTP que se deseja efectuar, que pode ser GET, POST, DELETE ou PUT.

As respostas retornadas de uma solicitação REST voltam sempre com um código de status HTTP que ajudam a verificar se a requisição foi feita com sucesso ou houve alguma inconsistência na execução da mesma, ajudando-nos, como desenvolvedores, a encontrar possíveis erros com mais rapidez.

* + 1. **Características**
* Utiliza os métodos HTTP (GET, POST, PUT, DELETE) para interagir com recursos.
* Baseado em URIs (Uniform Resource Identifiers).
* Usa formatos como JSON ou XML para representar dados.
* Vantagens:
* Simplicidade e ampla adoção.
* Compatibilidade com a maioria das linguagens e plataformas.
* Desvantagens:
* Pode ser menos flexível em comparação com GraphQL.
* Over-fetching de dados em algumas situações.
* Casos de Uso:
* Aplicações web e móveis.
  1. **WebSockets**

WebSockets é uma tecnologia avançada que torna possível abrir uma sessão de comunicação interativa entre o navegador do usuário e um servidor. Com esta API, você pode enviar mensagens para um servidor e receber respostas orientadas a eventos sem ter que consultar o servidor para obter uma resposta.

WebSocket é diferente do [HTTP](https://en.wikipedia.org/wiki/HTTP) usado para servir a maioria das páginas da Web. Embora sejam diferentes, [RFC](https://en.wikipedia.org/wiki/RFC_(identifier)) [6455](https://datatracker.ietf.org/doc/html/rfc6455) afirma que WebSocket "é projetado para trabalhar sobre as portas HTTP 443 e 80, bem como para suportar proxies HTTP e intermediários", tornando-o compatível com HTTP. Para obter compatibilidade, o [handshake](https://en.wikipedia.org/wiki/Handshake_(computing)) WebSocket usa o [cabeçalho HTTP Upgrade](https://en.wikipedia.org/wiki/HTTP/1.1_Upgrade_header)[[3]](https://en.wikipedia.org/wiki/WebSocket#cite_note-3) para mudar do protocolo HTTP para o protocolo WebSocket.

O protocolo WebSocket permite a interação [full-duplex](https://en.wikipedia.org/wiki/Duplex_(telecommunications)#Full_duplex) entre um [navegador da Web](https://en.wikipedia.org/wiki/Web_browser) (ou outro aplicativo [cliente](https://en.wikipedia.org/wiki/Client_(computing))) e um [servidor Web](https://en.wikipedia.org/wiki/Web_server) com menor sobrecarga do que alternativas half-duplex, como [sondagem](https://en.wikipedia.org/wiki/Polling_(computer_science)) HTTP, facilitando a transferência de dados em tempo real de e para o servidor. Isso é possível fornecendo uma maneira padronizada para o servidor enviar conteúdo para o cliente sem ser solicitado primeiro pelo cliente, e permitindo que as mensagens sejam passadas para frente e para trás enquanto mantém a conexão aberta. Dessa forma, uma conversa contínua bidirecional pode ocorrer entre o cliente e o servidor. As comunicações geralmente são feitas pela [porta](https://en.wikipedia.org/wiki/Port_(computer_networking)) TCP número 443 (ou 80 no caso de conexões não seguras), o que é benéfico para ambientes que bloqueiam conexões de Internet que não são da Web usando um [firewall](https://en.wikipedia.org/wiki/Firewall_(computing)). Além disso, o WebSocket habilita fluxos de mensagens sobre o TCP. O TCP sozinho lida com fluxos de bytes sem nenhum conceito inerente de uma mensagem. Comunicações bidirecionais semelhantes entre navegador e servidor foram alcançadas de maneiras não padronizadas usando tecnologias paliativas, como [o Comet](https://en.wikipedia.org/wiki/Comet_(programming)) ou [o Adobe Flash Player](https://en.wikipedia.org/wiki/Adobe_Flash_Player).

* + 1. **Características**
* Mantém uma conexão persistente.
* Suporta streaming de dados.
* Ideal para aplicativos em tempo real (chat, jogos, etc.).
  + 1. **Vantagens**
* Baixa latência e atualizações instantâneas.
* Eficiente para notificações em tempo real.
  + 1. **Desvantagens**
* Requer suporte no servidor e no cliente.
* Não é adequado para todas as situações.
* Casos de Uso:
* Aplicações de chat em tempo real, dashboards, etc.

1. **Arquiteturas para acesso remoto repositórios de dados** 
   1. **Arquitetura Monolítica**

A arquitetura monolítica é uma abordagem tradicional para o desenvolvimento de software, onde uma aplicação é construída como uma única unidade indivisível. Todos os componentes do sistema, como a interface do utilizador, lógica de negócio e acesso a dados, são combinados num único código-fonte e implantados como uma única aplicação executável.

A arquitetura monolítica é adequada para pequenas aplicações ou para projetos em fase inicial, onde a simplicidade e a velocidade de desenvolvimento são cruciais. No entanto, à medida que a aplicação cresce, os desafios de manutenção e escalabilidade podem justificar a migração para uma arquitetura mais modular, como microsserviços.

#### Características

* Coesão: Todos os componentes do sistema estão fortemente acoplados e são desenvolvidos, testados e implantados juntos.
* Facilidade de Desenvolvimento Inicial: No início, é mais simples desenvolver, testar e implantar uma aplicação monolítica devido à sua natureza unificada.
* Desempenho: Por funcionar como uma única unidade, a comunicação interna entre componentes é mais rápida, sem a necessidade de chamadas de rede.

#### Vantagens

* Simplicidade Inicial: A simplicidade de configuração e desenvolvimento inicial torna a arquitetura monolítica uma escolha atraente para pequenas equipas e projetos.
* Desempenho Superior: A comunicação entre componentes dentro de uma aplicação monolítica é mais eficiente comparada com sistemas distribuídos.
* Facilidade de Teste: Testar uma aplicação monolítica pode ser mais simples devido à ausência de complexidade na comunicação entre serviços.

#### Desvantagens:

* Escalabilidade Limitada: À medida que a aplicação cresce, torna-se difícil escalar componentes individualmente, podendo levar a problemas de desempenho e manutenção.
* Manutenção Complexa: A gestão de uma grande base de código monolítica pode tornar-se complexa e propensa a erros, dificultando a introdução de novas funcionalidades.
* Implantação Demorada: Qualquer alteração numa pequena parte do sistema requer a recompilação e a implantação de toda a aplicação, o que pode ser demorado e arriscado.

### **Arquitetura de Microsserviços**

A arquitetura de microsserviços é uma abordagem que estrutura uma aplicação como um conjunto de serviços pequenos e independentes, cada um executando uma funcionalidade específica e comunicando-se através de APIs.

#### Características:

* Desacoplamento: Cada serviço é desenvolvido, implantado e escalado de forma independente.
* Especialização: Os serviços são projetados para executar uma única função, permitindo uma especialização mais detalhada.
* Comunicação por API: A interação entre serviços é realizada por meio de APIs, frequentemente utilizando protocolos leves como HTTP/REST ou gRPC.

#### Vantagens:

* Escalabilidade: Cada microsserviço pode ser escalado individualmente, permitindo uma utilização mais eficiente dos recursos.
* Flexibilidade Tecnológica: Equipas diferentes podem utilizar diferentes tecnologias e linguagens de programação para desenvolver os seus serviços.
* Resiliência: Falhas num serviço específico não afetam diretamente outros serviços, aumentando a resiliência da aplicação.

#### Desvantagens:

* Complexidade: A gestão de múltiplos serviços independentes pode introduzir uma complexidade significativa em termos de implementação, teste e monitorização.
* Comunicação e Latência: A comunicação entre serviços pode introduzir latência e exigir um planeamento cuidadoso para garantir a eficiência.
* Consistência de Dados: Manter a consistência dos dados entre serviços distribuídos pode ser desafiador e pode requerer estratégias sofisticadas de gestão de dados.

A arquitetura de microsserviços é ideal para aplicações grandes e complexas que necessitam de alta escalabilidade e flexibilidade. Ela permite que equipas trabalhem de forma mais autónoma e que a aplicação cresça de maneira modular e escalável.

### **Arquitetura Serverless**

A arquitetura serverless é um modelo de execução em que o fornecedor de serviços em nuvem gerencia automaticamente a infraestrutura necessária para executar o código dos desenvolvedores. Os desenvolvedores escrevem funções que são invocadas em resposta a eventos, e o fornecedor de serviços em nuvem cuida do provisionamento, escalabilidade e gestão dos servidores.

#### Características:

* Execução sob Demanda: Funções são executadas em resposta a eventos específicos e são escaladas automaticamente.
* Gestão Automatizada de Infraestrutura: O fornecedor de serviços em nuvem gere toda a infraestrutura, permitindo que os desenvolvedores se concentrem apenas no código.
* Modelo de Pagamento por Utilização: Os custos são baseados na quantidade de recursos utilizados e no número de execuções, proporcionando uma eficiência de custos significativa.

#### Vantagens:

* Escalabilidade Automática: A infraestrutura é automaticamente escalada para atender à demanda, sem intervenção manual.
* Redução de Custos Operacionais: Os desenvolvedores pagam apenas pelo tempo de execução das suas funções, eliminando os custos associados a servidores ociosos.
* Aceleração do Desenvolvimento: Ao eliminar a necessidade de gerir a infraestrutura, os desenvolvedores podem concentrar-se em escrever código, acelerando o ciclo de desenvolvimento.

#### Desvantagens:

* Latência Inicial: A invocação inicial de uma função serverless pode introduzir uma latência adicional, conhecida como "cold start".
* Limitações de Tempo de Execução: Funções serverless geralmente têm limites de tempo de execução, o que pode ser inadequado para tarefas de longa duração.
* Dependência do Fornecedor: A utilização de serviços proprietários de um fornecedor específico pode criar uma dependência, dificultando a migração para outro fornecedor.

A arquitetura serverless é adequada para aplicações que têm cargas de trabalho variáveis e imprevisíveis, bem como para projetos que beneficiam de uma rápida iteração de desenvolvimento. Ela permite uma escalabilidade eficiente e redução de custos, embora seja necessário considerar os desafios associados à latência e à gestão de dependências de fornecedores.

1. **Conclusão**

Ao longo deste trabalho, exploramos diferentes modelos e arquiteturas de acesso remoto a repositórios de dados na web, nomeadamente SOAP, GraphQL, gRPC, REST e WebSockets. Cada um destes modelos oferece características únicas e vantagens específicas que podem ser aproveitadas dependendo do contexto e das necessidades da aplicação.

Compreendemos que as arquiteturas desempenham um papel crucial na forma como os dados são acessados, manipulados e transmitidos. As arquiteturas monolítica, de microsserviços e serverless, embora distintas, são fundamentais na concepção de sistemas modernos. A compreensão destas arquiteturas e modelos é essencial para qualquer desenvolvedor, pois permite escolher a abordagem mais adequada para cada projeto, equilibrando fatores como desempenho, escalabilidade, manutenção e custo. Este conhecimento possibilita a criação de sistemas mais eficientes e adaptáveis ao mundo digital de hoje.

1. **Referências bibliográficas**

* [**https://kondado.com.br/blog/blog/2023/08/02/o-que-e-um-repositorio-de-dados/**](https://kondado.com.br/blog/blog/2023/08/02/o-que-e-um-repositorio-de-dados/)
* [**https://cloud.google.com/learn/what-is-microservices-architecture?hl=pt-br**](https://cloud.google.com/learn/what-is-microservices-architecture?hl=pt-br)
* [**https://developer.mozilla.org/en-US/docs/Glossary/REST**](https://developer.mozilla.org/en-US/docs/Glossary/REST)
* [**https://learn.microsoft.com/pt-pt/dotnet/architecture/maui/accessing-remote-data**](https://learn.microsoft.com/pt-pt/dotnet/architecture/maui/accessing-remote-data)
* [**https://developer.mozilla.org/en-US/docs/Glossary/SOAP**](https://developer.mozilla.org/en-US/docs/Glossary/SOAP)
* **https://kmdtraveller.medium.com/**