



CEIA

CENTRO DE EXCELÊNCIA EM
INTELIGÊNCIA ARTIFICIAL



CEIA
CENTRO DE EXCELÊNCIA EM
INTELIGÊNCIA ARTIFICIAL

INF
INSTITUTO DE
INFORMÁTICA



FAPEG
Fundação de Amparo à Pesquisa
do Estado de Goiás

SECTI
Secretaria de
Estado de Ciência,
Tecnologia e
Inovação



EMBRAPII
Empresa Brasileira de Pesquisa
e Inovação Industrial

MINISTÉRIO DA
CIÊNCIA, TECNOLOGIA
E INOVAÇÃO
GOVERNO FEDERAL
BRASIL
UNIÃO E RECONSTRUÇÃO

Workshop MLOps 24/02

Arquitetura de Microsserviços & Criando APIs para RAG



Carlos Henrique



Pedro Saraiva

O QUE DEFINE UMA ARQUITETURA?

O QUE DEFINE UMA ARQUITETURA?

A arquitetura de software é a **organização estrutural fundamental de um sistema**, definindo componentes (módulos, serviços), suas propriedades externas e relacionamentos.

O QUE DEFINE UMA ARQUITETURA?

A arquitetura de software é a **organização estrutural fundamental de um sistema**, definindo componentes (módulos, serviços), suas propriedades externas e relacionamentos.

Ela estabelece as bases técnicas e as diretrizes de design que **orientam o desenvolvimento**, garantindo que o software seja **escalável, seguro e fácil de manter**.

O QUE DEFINE UMA ARQUITETURA?

A arquitetura de software é a **organização estrutural fundamental de um sistema**, definindo componentes (módulos, serviços), suas propriedades externas e relacionamentos.

Ela estabelece as bases técnicas e as diretrizes de design que **orientam o desenvolvimento**, garantindo que o software seja **escalável, seguro e fácil de manter**.

Exemplos de Arquiteturas de Software

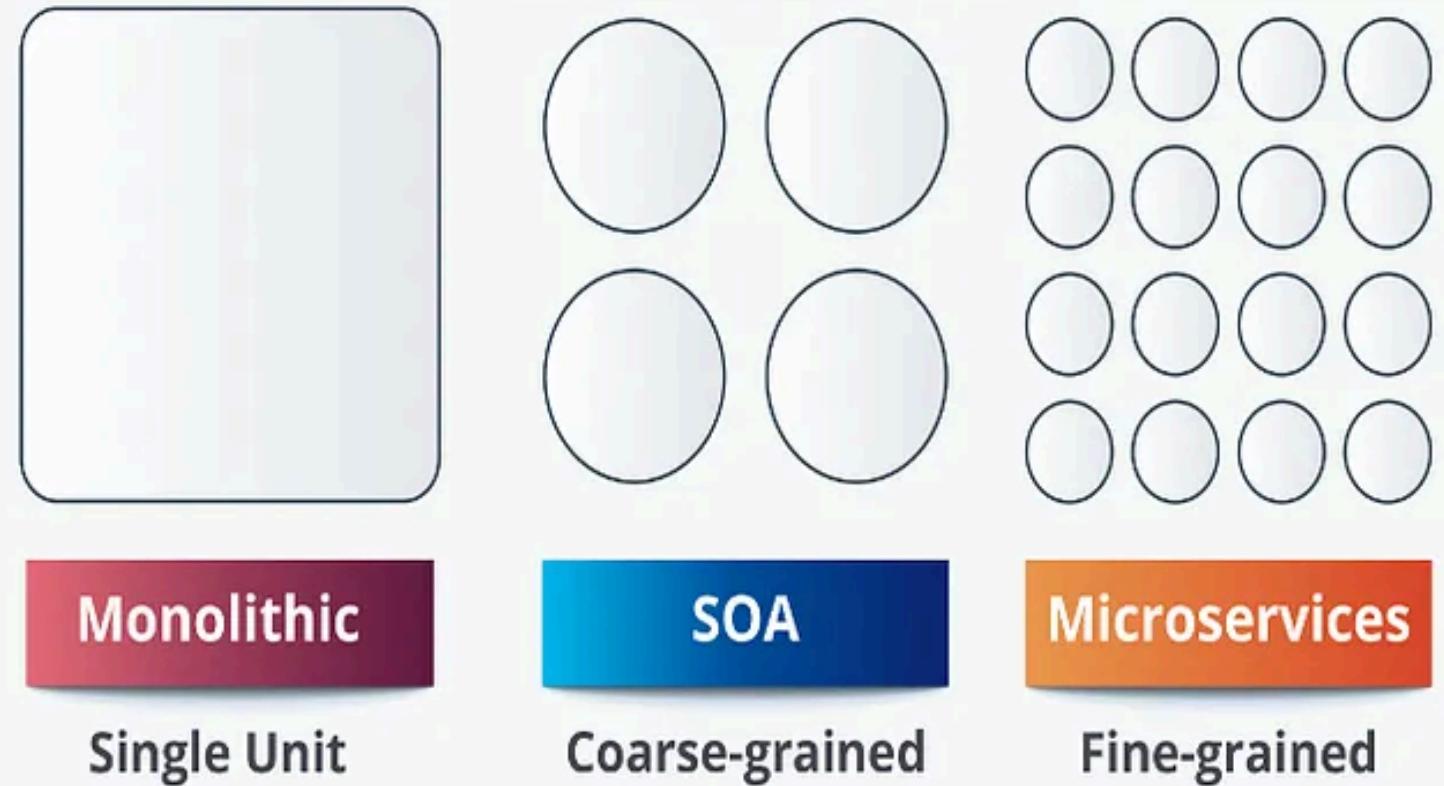
- Modelo cliente-servidor
- Sistema de processamento distribuído
- Peer-to-Peer (P2P)
- Model-View-Controller (MVC)
- **Aplicação monolítica**
- **Arquitetura Orientada à Serviços**
- **Arquitetura de Microsserviços**

ARQUITETURA DE SERVIÇOS

Monolítico: é semelhante a um grande contêiner onde todos os componentes de software de um aplicativo são montados e compactados.

SOA (Arquitetura Orientada à Serviços): é essencialmente uma coleção de serviços. Esses serviços se comunicam entre si. A comunicação pode envolver uma simples passagem de dados ou dois ou mais serviços coordenando alguma atividade.

Monolithic vs. SOA vs. Microservices

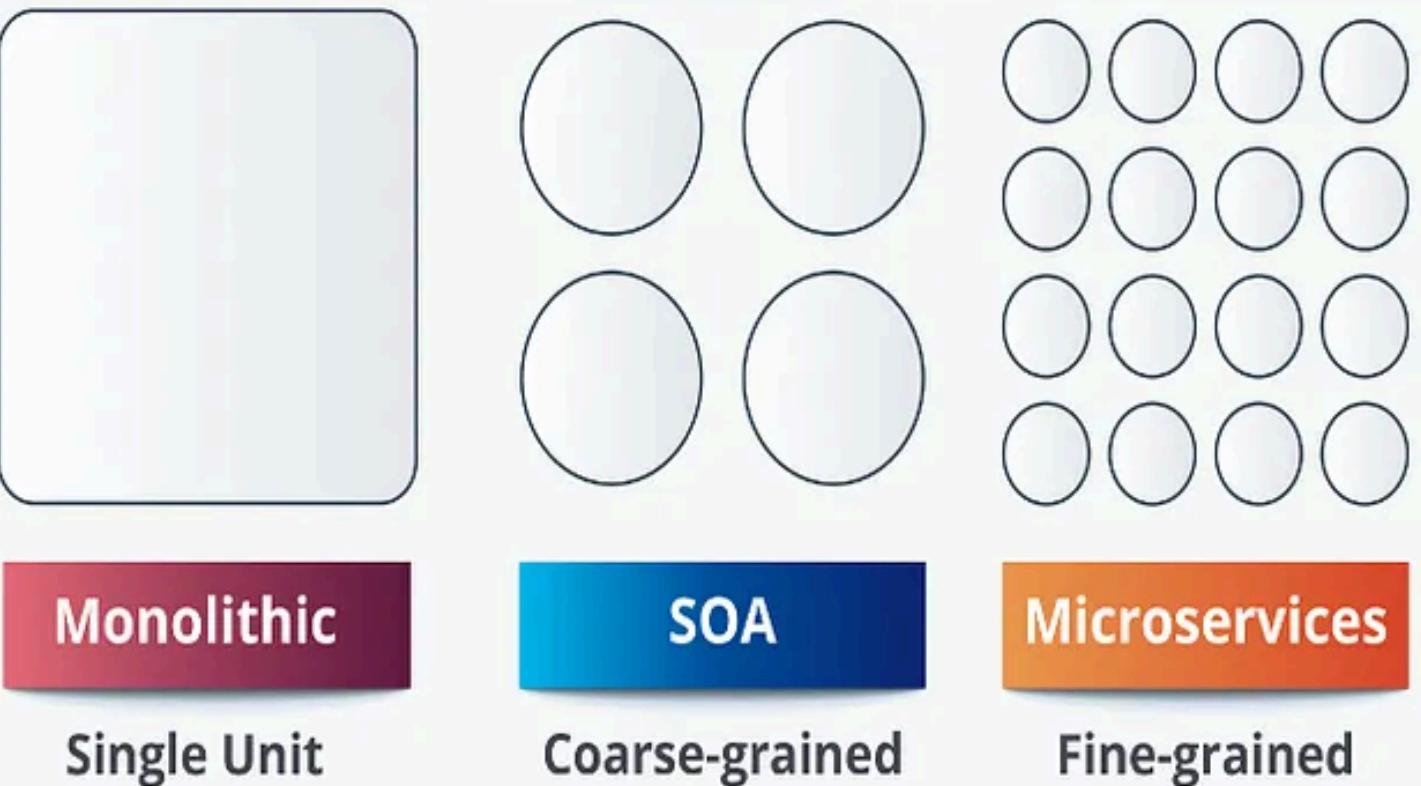


ARQUITETURA DE SERVIÇOS

Arquitetura de Microsserviços: é um estilo arquitetônico que estrutura um aplicativo como uma coleção de pequenos serviços autônomos modelados em torno de um domínio de negócios.

Enquanto que o SOA é **Enterprise-wide** (focado em fazer várias aplicações ou serviços diferentes de um negócio se comunicarem), a arquitetura de microsserviços trabalha em cima da mesma aplicação, dividindo seus componentes em pequenos serviços que se comunicam entre si.

Monolithic vs. SOA vs. Microservices

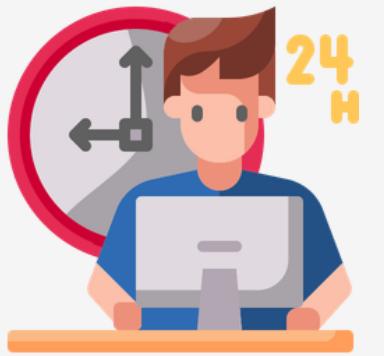


O QUE É UM MICROSERVIÇO?

O QUE É UM MICROSERVIÇO?



**Microsserviço responsável
por interagir com o BD**



**Microsserviço trabalhador
(worker)**



Microsserviço de API



**Microsserviço de
Mensageria**

AUTONÔMO



ESPECIALIZADO



Cada serviço do componente de uma arquitetura de microserviços pode ser **desenvolvido, implantado, operado e escalado sem afetar o funcionamento de outros serviços.**

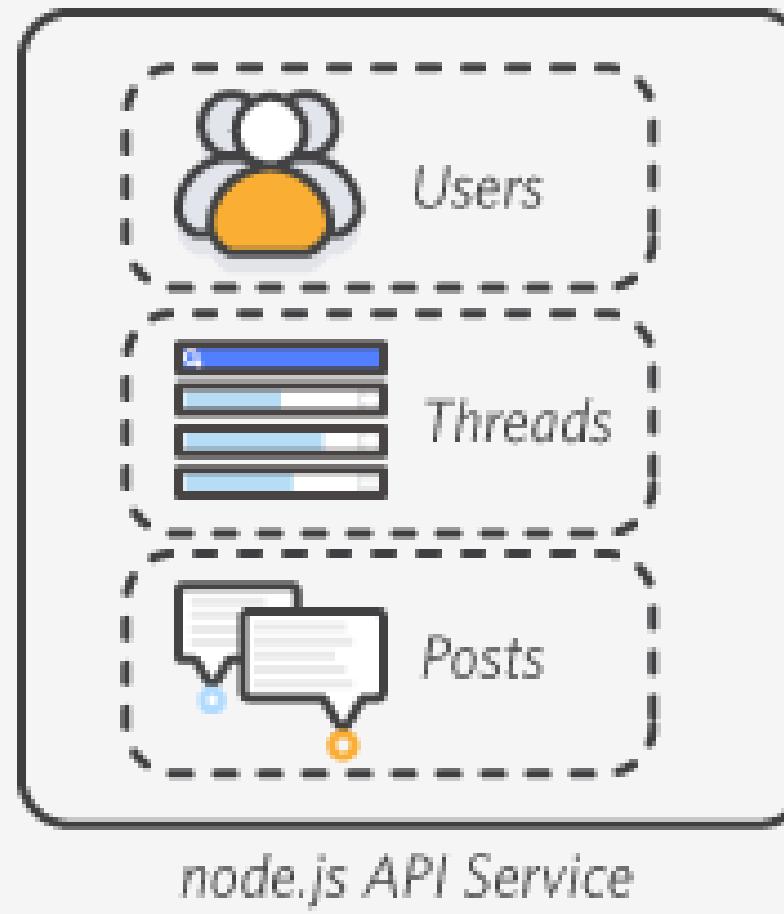
Os serviços não precisam compartilhar nenhum código ou implementação com os outros serviços. Todas as comunicações entre componentes individuais ocorrem por meio de APIs bem definidas.

Cada serviço é projetado para ter um conjunto de recursos e é **dedicado à solução de um problema específico.**

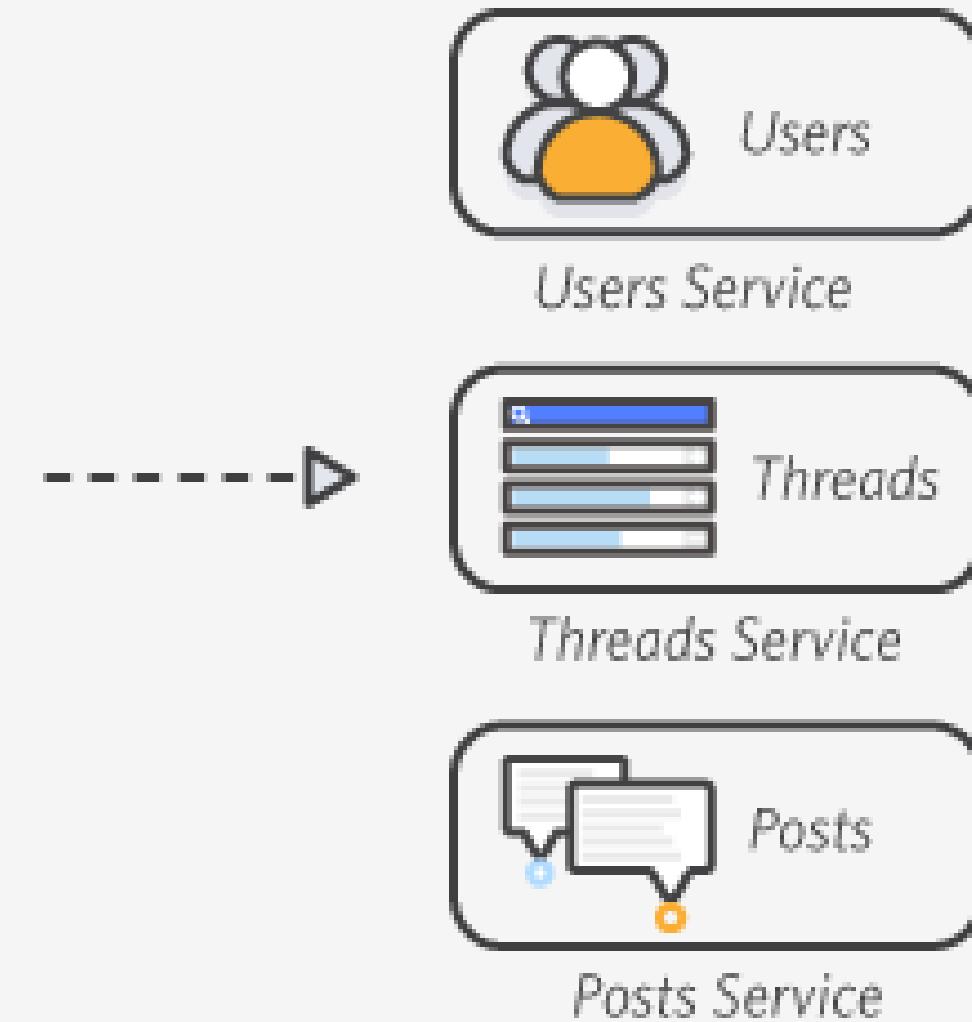
Se os desenvolvedores acrescentarem mais código a um serviço ao longo do tempo, aumentando sua complexidade, ele poderá ser dividido em serviços menores.

O QUE É UM MICROSERVIÇO?

1. MONOLITH



2. MICROSERVICES



**Quais boas práticas tomar ao se implementar
uma arquitetura de microsserviços em um
processo de MLOps?**

Quais boas práticas tomar ao se implementar uma arquitetura de microserviços em um processo de MLOps?



**Desenvolvimento
& Entrega**



Empacotamento

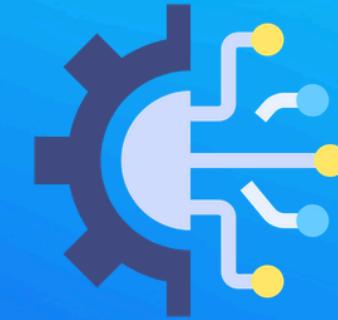


**Cointainerização &
Gerenciamento**



**Escalabilidade &
Desempenho**

DEVELOPMENT & DEPLOYMENT



Uma boa prática é criar um modelo otimizado off-line, que pode então ser validado offline em um conjunto de dados de validação predefinido e depois carregado na arquitetura de microsserviços.

Os modelos devem ser **fáceis** de desenvolver e **fáceis** de se entregar. Além disso, é recomendado utilizar um serviço de mensageria para colher dados de features críticos.

PACKAGING



A arquitetura de microsserviços permite duas abordagens de publicações:

- Um microsserviço **standalone de ML**.
- Um microsserviço com **funcionalidades de aplicação + capacidade de ML**.

Em ambos os casos, é recomendado gerenciar separadamente os componentes de ML devido às suas complexidades (aquisição de dados, treinamento/desenvolvimento e atualização).

CONTAINERIZATION & MANAGEMENT



SCALING & PERFORMANCE



Duas etapas para integrar o algoritmo de ML ao resto da aplicação:

Disponibilizar o serviço de ML via uma **API REST**. A **API**, o **runtime** e o **algoritmo** precisam ser empacotados em um container.

Todos os microsserviços (containers) precisam ser gerenciados e orquestrados de forma efetiva (Kubernetes).

Como a **nuvem** e os **containers** servem como base da arquitetura de microsserviços, a escalabilidade e o desempenho se tornam aspectos integrais.

Ao selecionar e desenvolver diferentes componentes da aplicação, é necessário escolher cuidadosamente bancos de dados, runtimes, armazenamento e etc. com base no **desempenho** e na **escalabilidade** desejados.

Padrão de Microsserviço para componentes de RAG

Conhecemos as boas práticas e o fluxo de uma arquitetura de **microsserviços em MLOps**. Mas, como criamos uma? O primeiro passo é **identificar os componentes de um serviço**.

Para um sistema escalável de **RAG**, podemos decompor nos seguintes serviços:

- **Query Preprocessing Service.** Responsável por tarefas como normalização de consultas, correção ortográfica, detecção de intenções, expansão de consultas e etc. Este serviço garante que a **entrada para o estágio de recuperação seja otimizada**.
- **Retrieval Service(s).** Este é o componente central. Pode ser um único serviço ou decomposto ainda mais:
 - **Dense Retrieval Service.** Interage com bancos de dados vetoriais para encontrar documentos semanticamente semelhantes.
 - **Sparse Retrieval Service.** Utiliza técnicas tradicionais de Information Retrieval (IR) que usa vetores esparsos para buscas **keyword-based**.
 - **Hybrid Search Orchestrator Service.** Combina o resultado dos **Dense Retrieval** e **Sparse Retrieval**, abstraindo a complexidade da pesquisa.

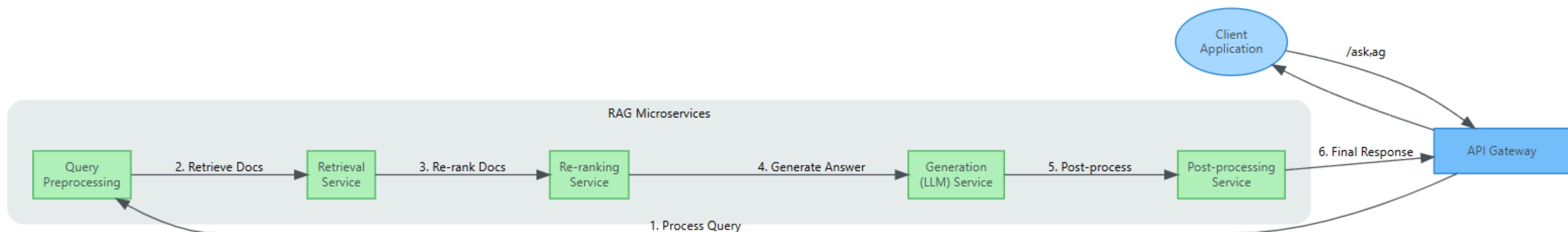
Padrão de Microsserviço para componentes de RAG

- **Re-ranking Service.** Pega o conjunto inicial de documentos recuperados e aplica modelos mais sofisticados, muitas vezes computacionalmente intensivos (ex. cross-enconders) para melhorar a classificação de relevância.
- **Context Aggregation and Formatting Service.** Reúne os documentos reclassificados, extrai passagens relevantes e os formata em uma sequência de contexto coerente adequada para o LLM, gerenciando potencialmente os limites de token.
- **Generation Service (LLM Abstraction Service).** Este serviço encapsula interações com um ou mais LLM. Ele lida com engenharia imediata, chamadas de API para endpoints de atendimento LLM e gerencia configurações.
- **Post-processing Service.** Processa a saída do LLM. Isso pode incluir a geração de citações, a realização de verificações de segurança, a filtragem de conteúdo prejudicial ou a formatação da resposta final para o usuário.

Padrão de Microsserviço para componentes de RAG

- **Embedding Service.** Esse serviço é responsável por gerar embeddings em tempo real (online). Embora, normalmente, seja feita em batch (offline) para o corpus principal, esse serviço cuida de cenários que exigem **execução imediata** durante a inferência ou ingestão dinâmica.
- **Feedback and Logging Service.** Um serviço dedicado para coletar feedback do usuário, explícito ou implícito, e para agregar logs de outros serviços para monitoramento e análise.

- **API Gateway.** Um API Gateway atua como um ponto de entrada único para todas as solicitações do cliente ao seu sistema RAG. Em vez de clientes chamarem microsserviços individuais diretamente, eles enviam solicitações ao API Gateway. O gateway então encaminha essas solicitações para os microsserviços downstream apropriados.



Padrões de Design de Microsserviços para Sistema RAG

- 01** Enhanced Fault Isolation
- 02** Reduce Technology or Vendor Lock-in
- 03** User Friendly
- 04** Smaller and Quicker Deployments
- 05** Scalability

Benefícios da Arquitetura de Microsserviços

Muito obrigado!

**Agora para a prática
e dúvidas...**