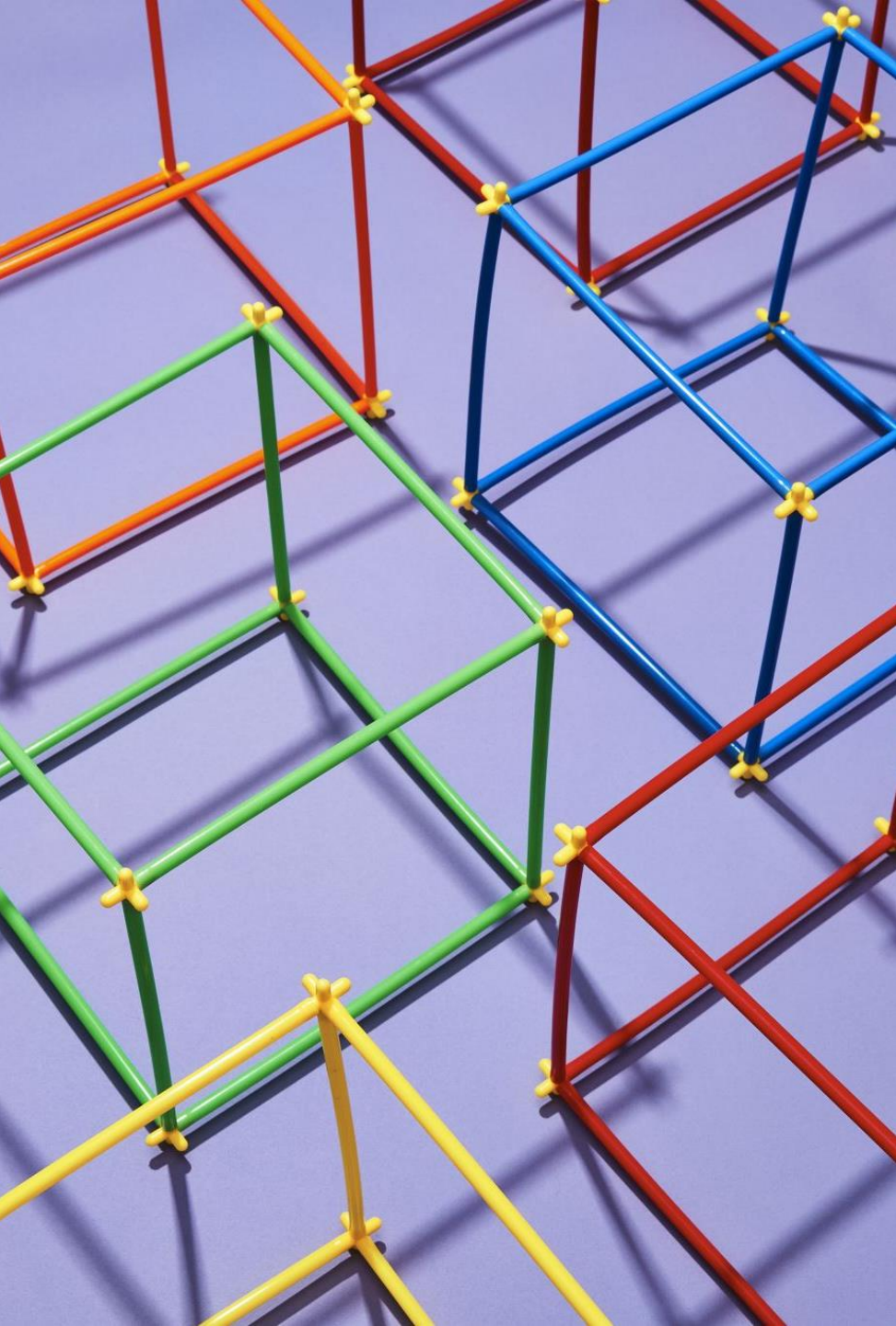


A 3D isometric illustration of a person with brown hair wearing a white headset and a light green long-sleeved shirt, sitting in a white office chair at a pink desk. The person is typing on a keyboard. On the desk, there is a computer monitor displaying a simple interface, a yellow mug, a smartphone, and a small blue book. Above the desk, there are two pink shelves. The top shelf holds a yellow desk lamp, a small potted plant, a red-framed photo of two people, and two books. The bottom shelf holds a small blue clock and a dark green book. To the right of the desk, there is a tall green cactus in a yellow pot. The desk is on a yellow rug with pink fringe. The background is a solid light pink color.

Microserviços – Arquitetura de Aplicações

Thiago Rodrigues

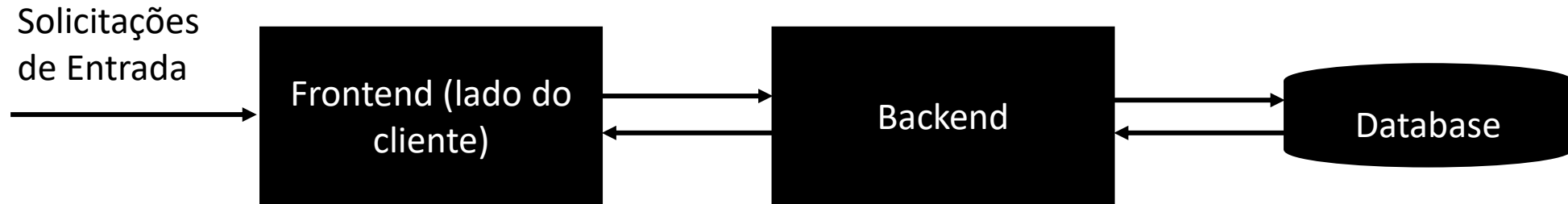


Agenda

- Arquitetura em Três Camadas
- Arquitetura Monolítica
- Arquitetura de Microserviços
- Vantagens e Desvantagens
- Padrões em Microserviços
- Serviço de Descoberta
- Registro de Serviços
- API Gateway

Arquitetura em Três Camadas

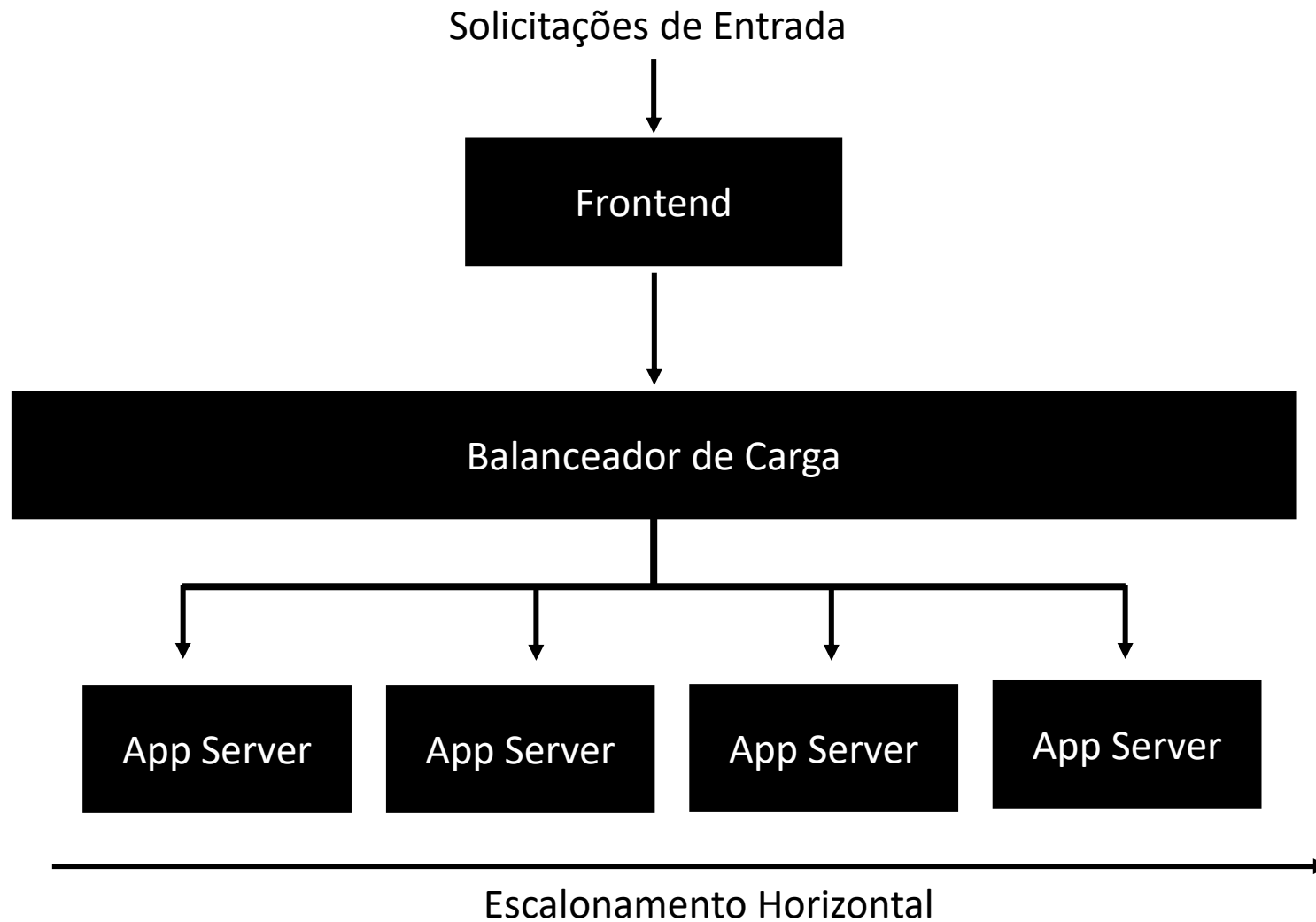
- Quase todas as aplicações de software escritas atualmente podem ser divididas em três elementos distintos:
 - Frontend
 - Backend
 - Armazenamento de dados



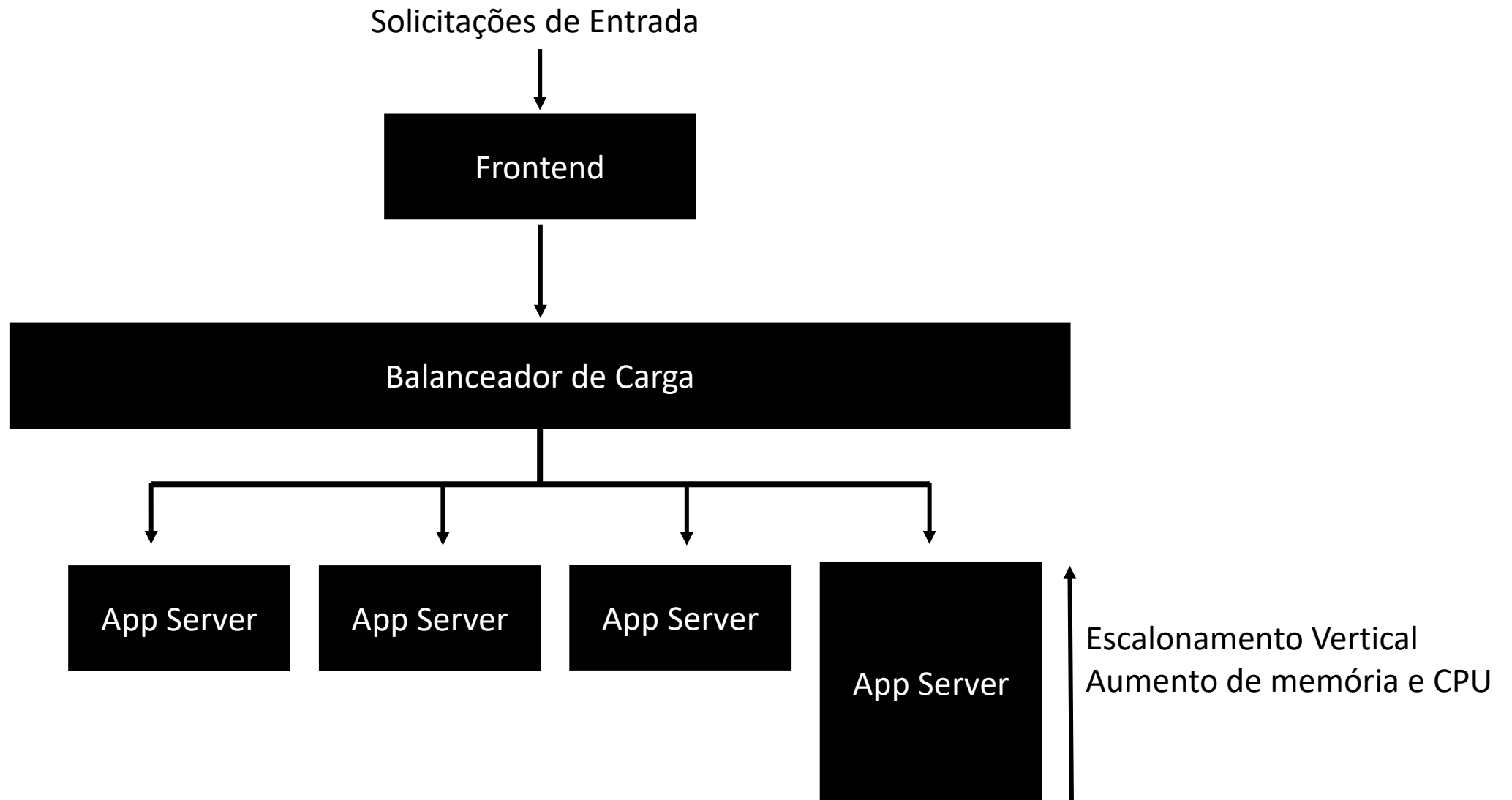
Arquitetura em Três Camadas

- Existe três maneiras diferentes de combinar esses elementos para criar uma aplicação.
 - Frontend e Backend na mesma aplicação e banco de dados separado
 - Frontend e Backend como aplicações diferentes, acompanhados por um banco de dados externo
 - Frontend e Backend na mesma aplicação com armazenamento em memória.

Dimensionamento Horizontal



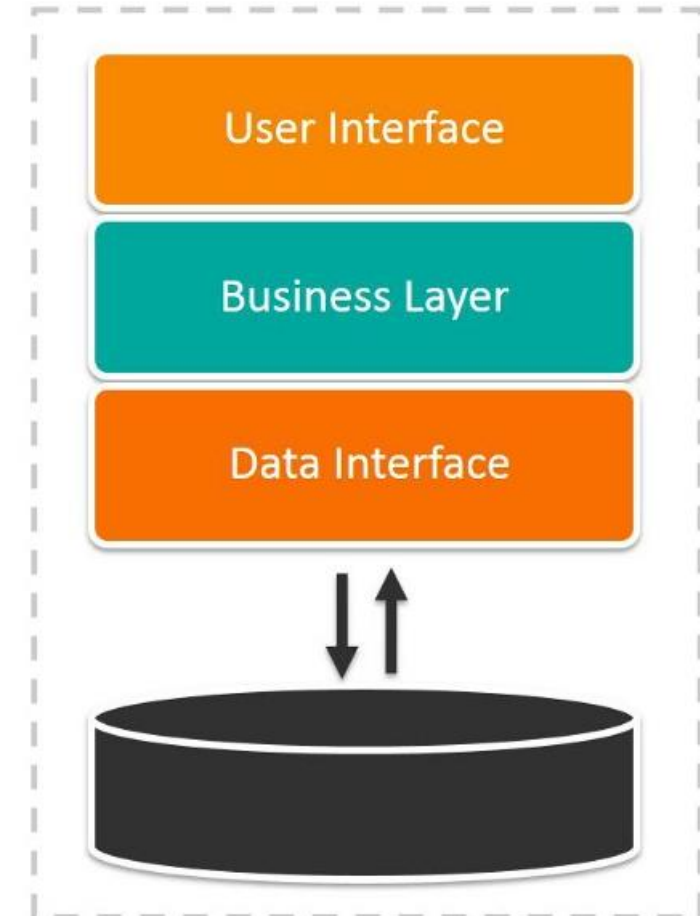
Dimensionamento Vertical



Arquitetura Monolítica

- Arquitetura Monolítica é um sistema único, não dividido, que roda em um único processo, uma aplicação de software em que as camadas de arquitetura dependem umas das outras.

Monolithic Architecture

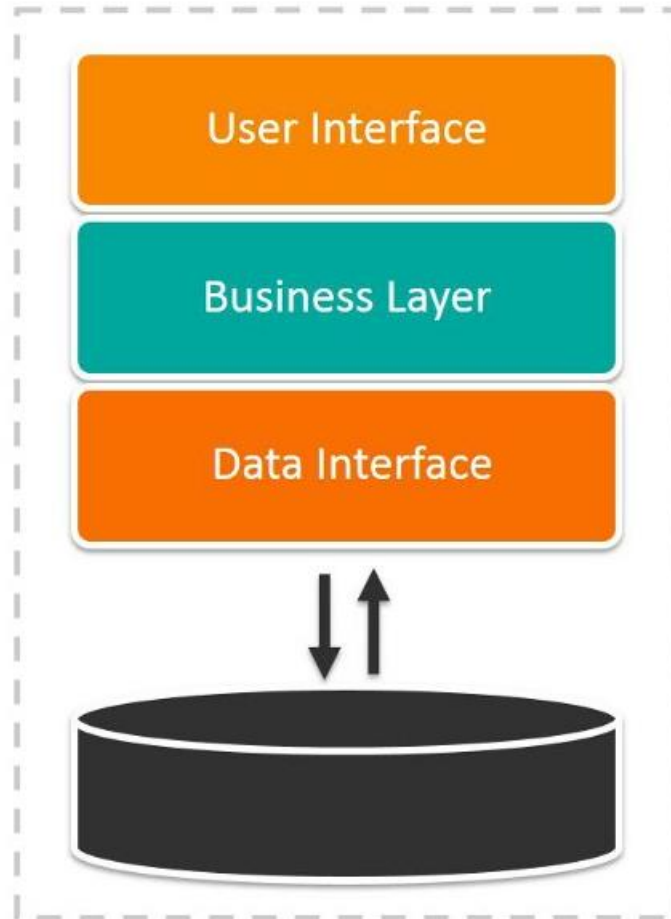


Tipos de Sistemas Monolíticos

- Sistema Monolítico de um só Processo
- Sistema Monolítico Modular

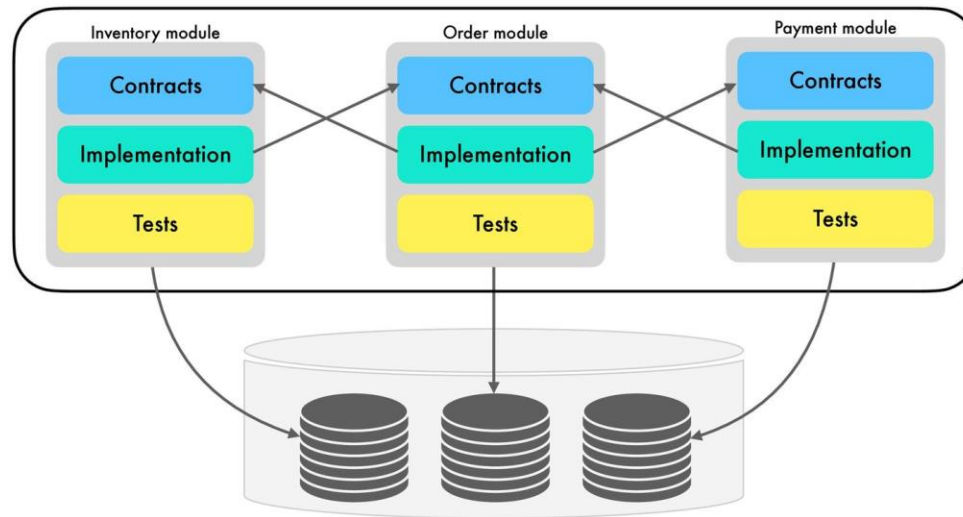
Sistema Monolítico de um só Processo

Monolithic Architecture



Sistema Monolítico Modular

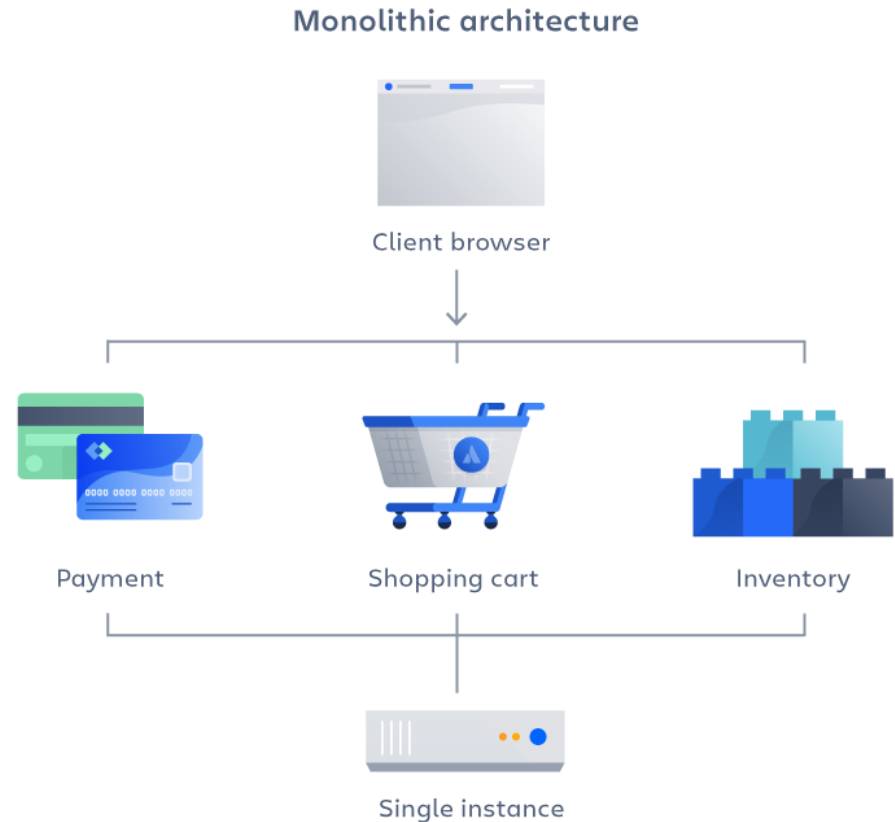
MODULAR MONOLITH



- A troca de informações entre os módulos ocorre dentro de um mesmo processo.

Arquitetura Monolítica

- Como exemplo, imagine a criação de uma aplicação para uma loja de venda de produtos, onde há os setores de **estoque**, **pagamento** e **carrinho**.
- Todos os usuários utilizarão o mesmo sistema, e será preciso que ele esteja dividido em algumas partes, são elas:
 - Autenticação e perfis de usuários (administração, contabilidade, estoque, vendedor);
 - Gráficos para a administração com os dados diários da loja;
 - Compra de produtos;
 - Estoque;
 - Vendas.



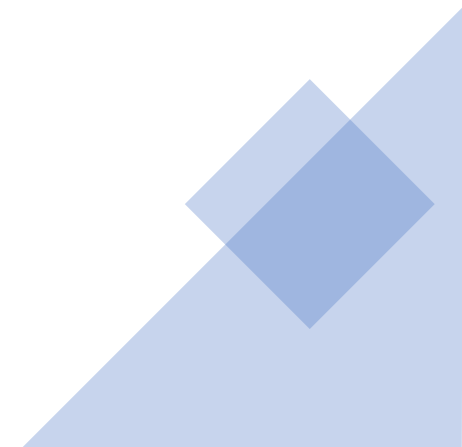
Observações

- Mais vulneráveis aos perigos do acoplamento.
- Quanto mais pessoas estiverem trabalhando em um mesmo lugar, elas começarão a atrapalhar umas às outras.
- Desenvolvedores diferentes querendo alterar a mesma parte do código.
- Desavença nas entregas.
 - Desafios das linhas confusas de responsabilidade

Arquitetura Monolítica não deve ser vista como algo que deve ser Evitado



Arquitetura Monolítica

- **Vantagens**
 - **Mais simples de desenvolver (código):** a organização fica concentrada em um único sistema;
 - **Simple de testar:** é possível testar a aplicação de ponta a ponta em um único lugar;
 - **Simple de fazer o deploy para o servidor:** a alteração é simplesmente feita e pronto;
- 

Arquitetura Monolítica

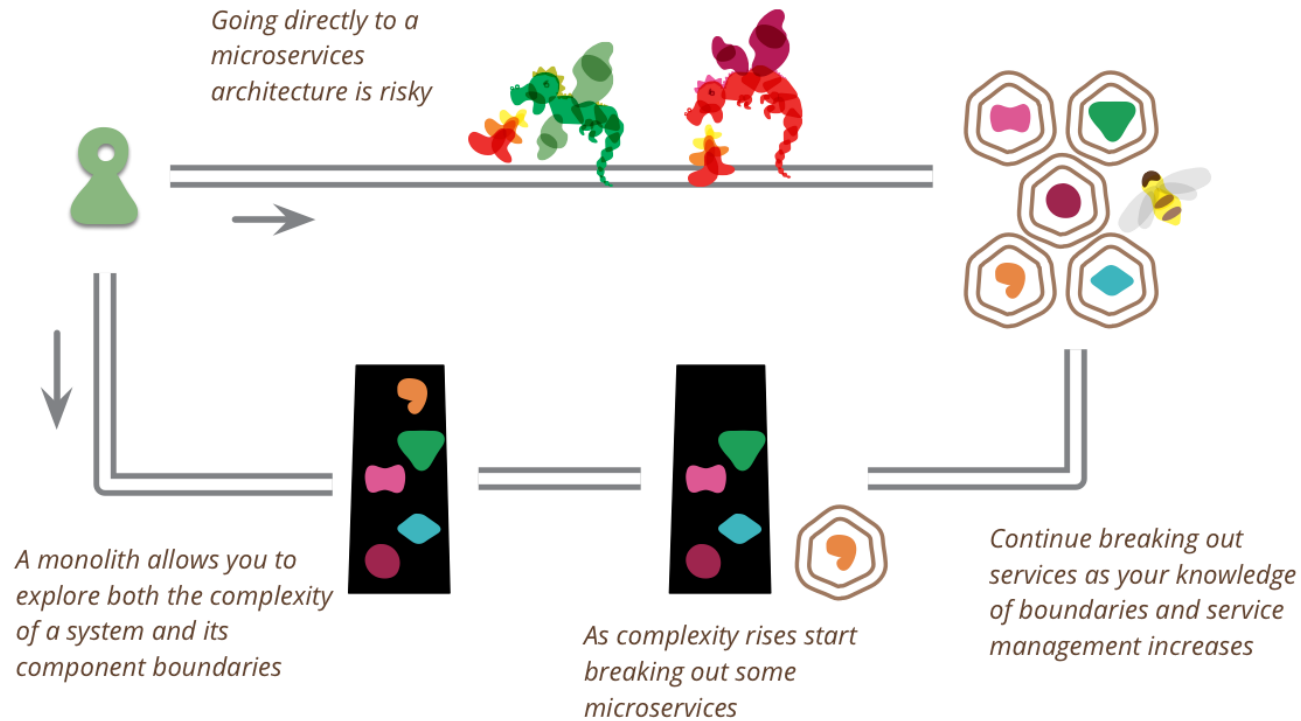
- **Desvantagens**

- **Baixa Escalabilidade:** grande base de código;
- **Manutenção:** a aplicação se torna cada vez maior de acordo com o seu tamanho, o código será cada vez mais difícil de entender e o desafio de fazer alterações rápidas e ter que subir para o servidor só cresce;
- **Alterações:** para cada alteração feita, é necessário realizar um novo deploy de toda a aplicação;
- **Linha de código:** uma linha de código que subiu errada pode quebrar todo o sistema e ele ficar totalmente inoperante;
- **Linguagens de programação:** não há flexibilidade em linguagens de programação. Aquela que for escolhida no início do projeto terá que ser seguida, sempre. Se o desenvolvimento de uma nova funcionalidade exigir outra linguagem de programação, existem duas possibilidades: ou todo o código é alterado ou a arquitetura do sistema precisará ser trocada.

Monolito vs Microserviços

- **Questões Importantes** (Segundo [Martin Fowler](#))
 - Quase todas as histórias bem-sucedidas de microserviços começaram com um monólito que ficou muito grande e foi quebrado
 - Quase todos os casos em que ouvi falar de um sistema que foi construído como um sistema de microserviços do zero acabaram em sérios problemas.

Monolith First



- <https://martinfowler.com/bliki/MonolithFirst.html>

A maneira lógica é projetar um monólito com cuidado, prestando atenção à modularidade dentro do software, tanto nos limites da API quanto em como os dados são armazenados.

Microserviços

- *Microserviços são serviços que podem ser implantados de forma independente, e são modelados em torno de uma modelo de negócio.*
- Implantações Independentes
- Modelagem em torno de um domínio de negócio
 - Domain Driven Design
- Ser responsável pelos seus próprios dados



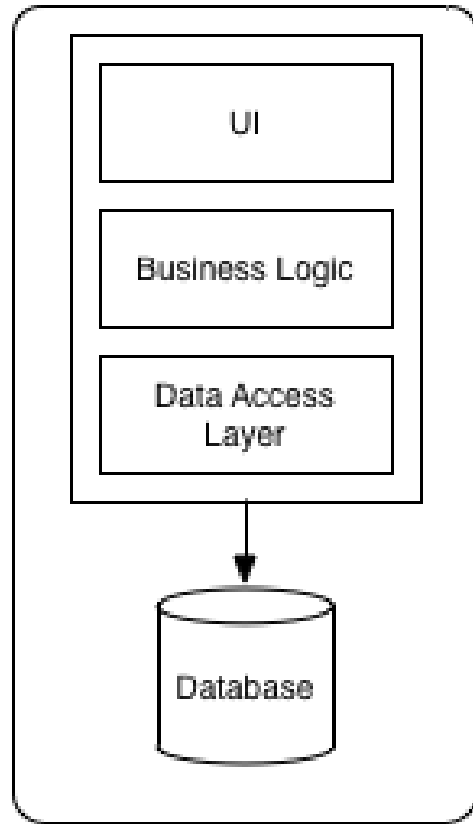
Microserviços

- *O Estilo arquitetural de microserviços é uma abordagem para desenvolver uma única aplicação como uma suíte de pequenos serviços, cada qual rodando em um processo próprio e se comunicando através de mecanismos leves, frequentemente através de APIs HTTP. Esses serviços são construídos através de pequenas responsabilidades e publicados independentemente através de um processo de deploy automatizado. **By Martin Fowler and James Lewis***

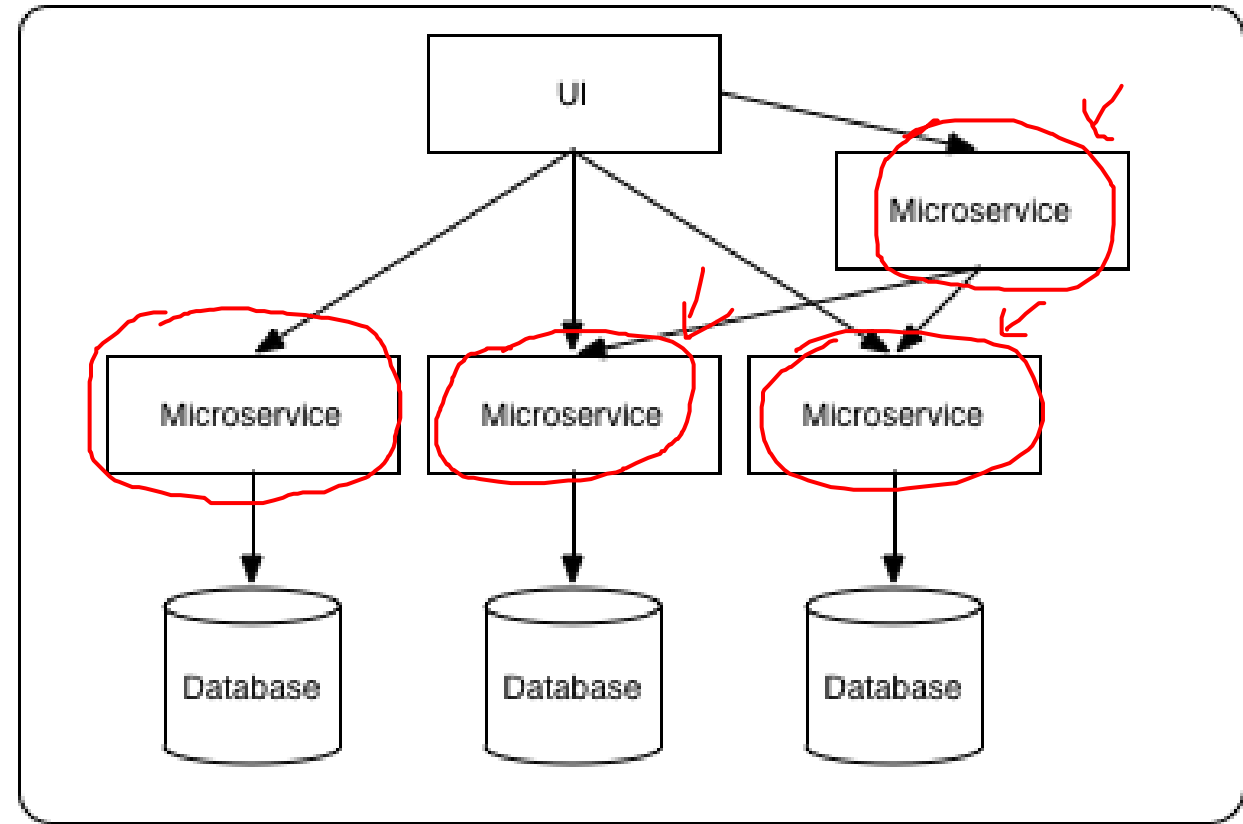
Características

- Descentralizados
 - Deploy rápido e sem **dependências**
 - Bancos de dados únicos para cada serviço
- Políglotas
- Autônomos
- Princípio da Responsabilidade Única
 - Microsserviços devem fazer uma única coisa e fazer bem

Arquitetura Micro Serviços



Monolithic Architecture



Microservices Architecture



Vantagens

- Deploys pequenos e mais rápidos: Fácil de fazer implantações em qualquer ambiente.
- Fácil de Entender
- Escalabilidade
- Alinhamento Organizacional
- Resiliência
 - Isolar componentes quando começar falhas
- Plataforma e linguagens agnósticas
- Menor Risco na adoção de novas tecnologias

Desvantagens

- Difícil de Testar
- Difícil de Monitorar
- Dificuldade de encontrar problemas
- Comunicação mais complexa
- Implementação de Segurança nas APIs



Padrões em Microserviços

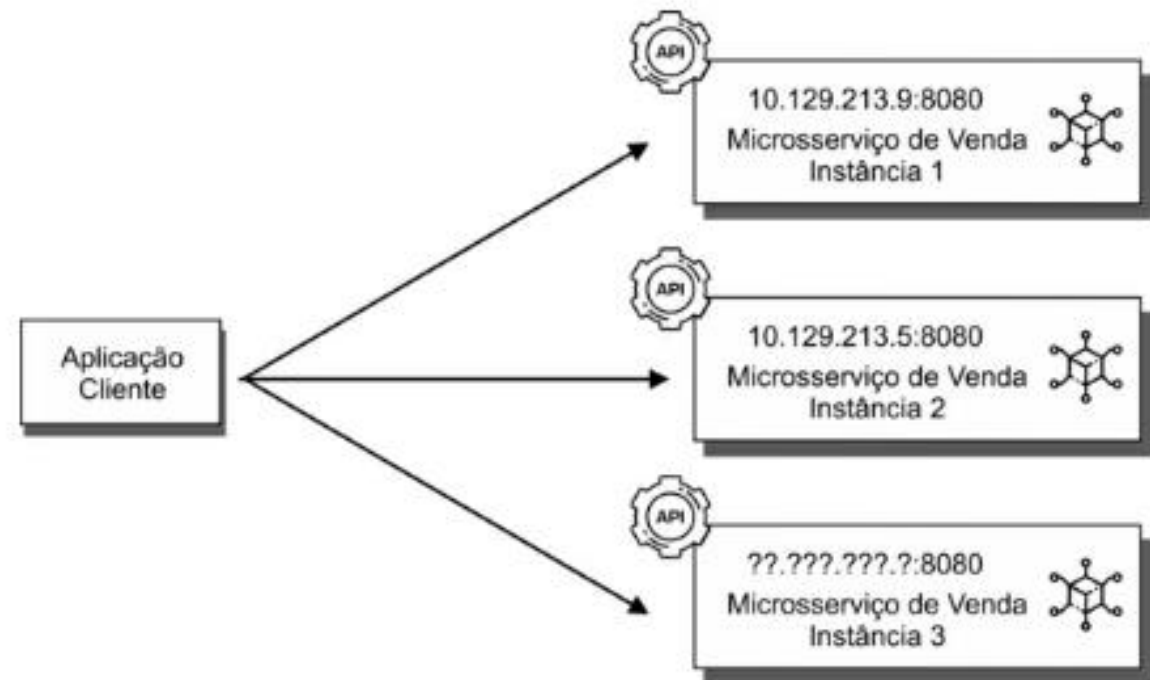
- Microserviços são modelados para serem isolados e independentes, atendendo a um subdomínio específico.
- Mas irá existir momento que eles precisaram se **comunicar** entre si.

Serviço de Descoberta (Service Discovery)

- Para que os microsserviços **interajam** entre si **através** de **APIs REST**, eles precisam **conhecer** os **endereços** de rede (IP e porta) **uns dos outros**, e o serviço de descoberta (**service discovery**) desempenham papel fundamental nesse processo.
- O **service discovery** é um **padrão** que busca solucionar o problema de **mapeamento de endereços** de rede quando este é feito de **forma dinâmica** e pode ser **implementado** tanto na **aplicação cliente**, que inicia a requisição para um outro microsserviço, como em um terceiro componente conhecido como roteador (router).

Serviço de Descoberta (Service Discovery)

- Endereços Dinâmicos.
- Gerenciados pelo provedor de serviço de cloud.
- Autoscale (Microserviço sendo atualizado automaticamente).



Registro de Serviço

- O **registro de serviços** (service registry é a estrutura central do service discovery e, de forma simplificada, pode ser definido como o **banco de dados** onde são **armazenados os mapeamentos** entre um **microserviço** e **seus endereços de rede** (IP e porta) dentro do cluster.
- **Características (Teorema CAP)**
 - Altamente Disponível
 - Qualquer cliente que requisitar um dado obterá resposta
 - Consistente
 - Clientes veem ao mesmo dado ao mesmo tempo
 - Partição Tolerante a Falhas
 - O sistema deverá continuar executando mesmo se ocorrer uma ou mais falhas

Exemplos

- Netflix Eureka
- Apache ZooKeeper
- Consul
- ETCD

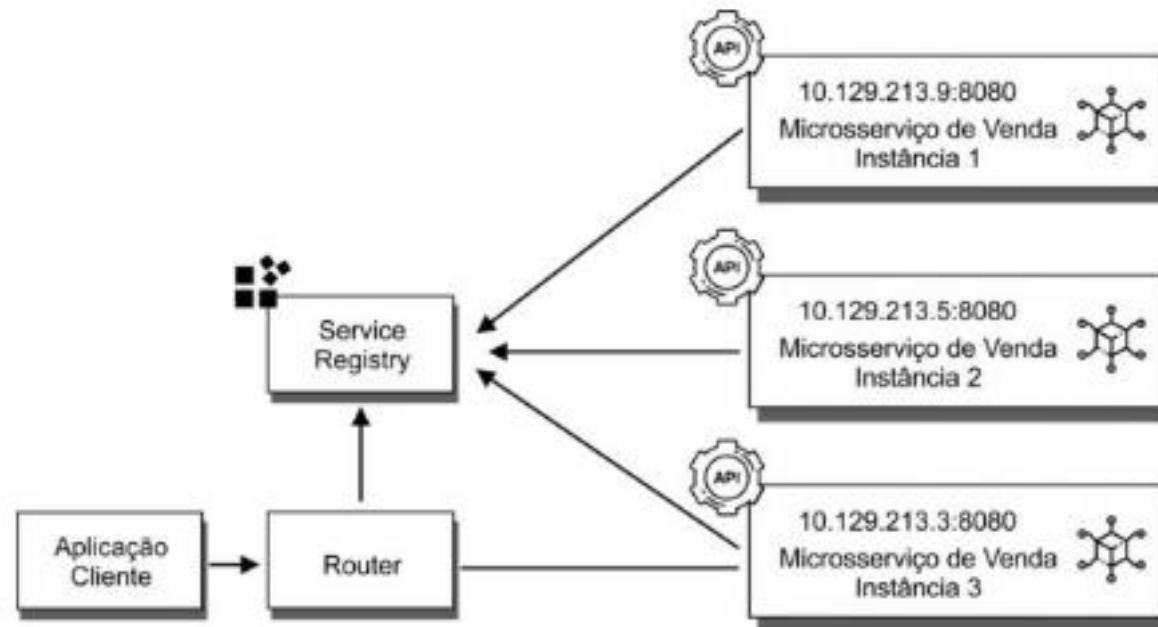
Registro de Serviço

- O seu funcionamento se baseia em **armazenar** no **service registry** o **mapeamento** do endereço de rede quando alguma **nova instância** de algum **microserviço** é avaliada como **disponível** e **operacional** e, também, em **remover** esse mapeamento quando assim for pertinente.

Descoberta de Serviço Via Router


- Uma das formas de **consumir** os **dados armazenados** por um service registry é tendo uma outra aplicação, denominada **router** (roteador), que deliberadamente **possui endereço estático** e conhecido, normalmente também **exposto via DNS**.
- O **router** fica responsável por **descobrir o endereço** de rede **consultando o service registry**, que **retorna uma lista de endereços** e distribui a carga entre eles.

Descoberta de Serviço Via Router






Descoberta de Serviço Via Router

- A vantagem desse modelo é desacoplar dos clientes os detalhes de implementação de descoberta dos serviços interagindo com o service registry, mantendo apenas um ponto de entrada no router e evitando múltiplas implementações da lógica de distribuição de carga em todos os microserviços.
 - Outra grande vantagem é que esse padrão é transparente e implementado por diversas soluções de cloud e orquestradores de serviços.
- 



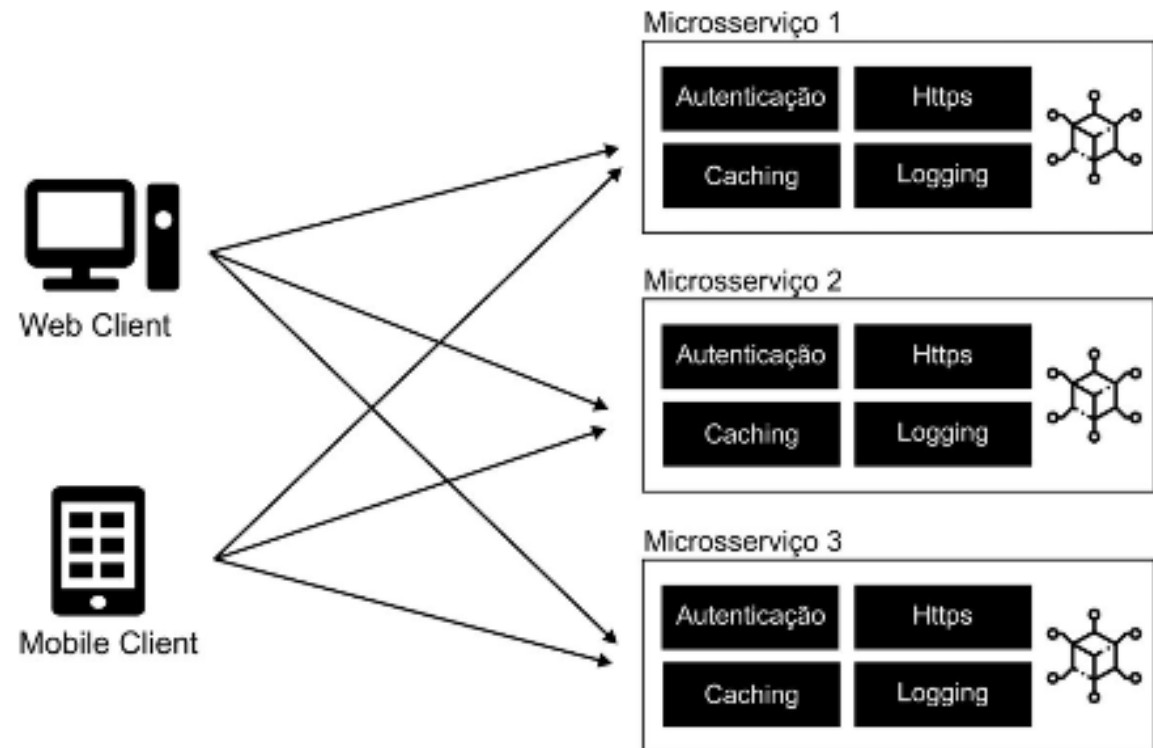
Padrão de Composição

- **Complexidades da arquitetura de Microserviços**
 - Alta granularidade de APIs
 - Comunicação com serviços em diferentes linguagens
 - Diferentes clientes acessando a mesma API
 - **Padrão API Gateway**
 - Centralizar as chamadas em único ponto de acesso, facilitando a segurança, o controle de métricas, o tratamento de cache, log e protocolos e outras questões que funcionam melhor de forma centralizada
- 

API Gateway

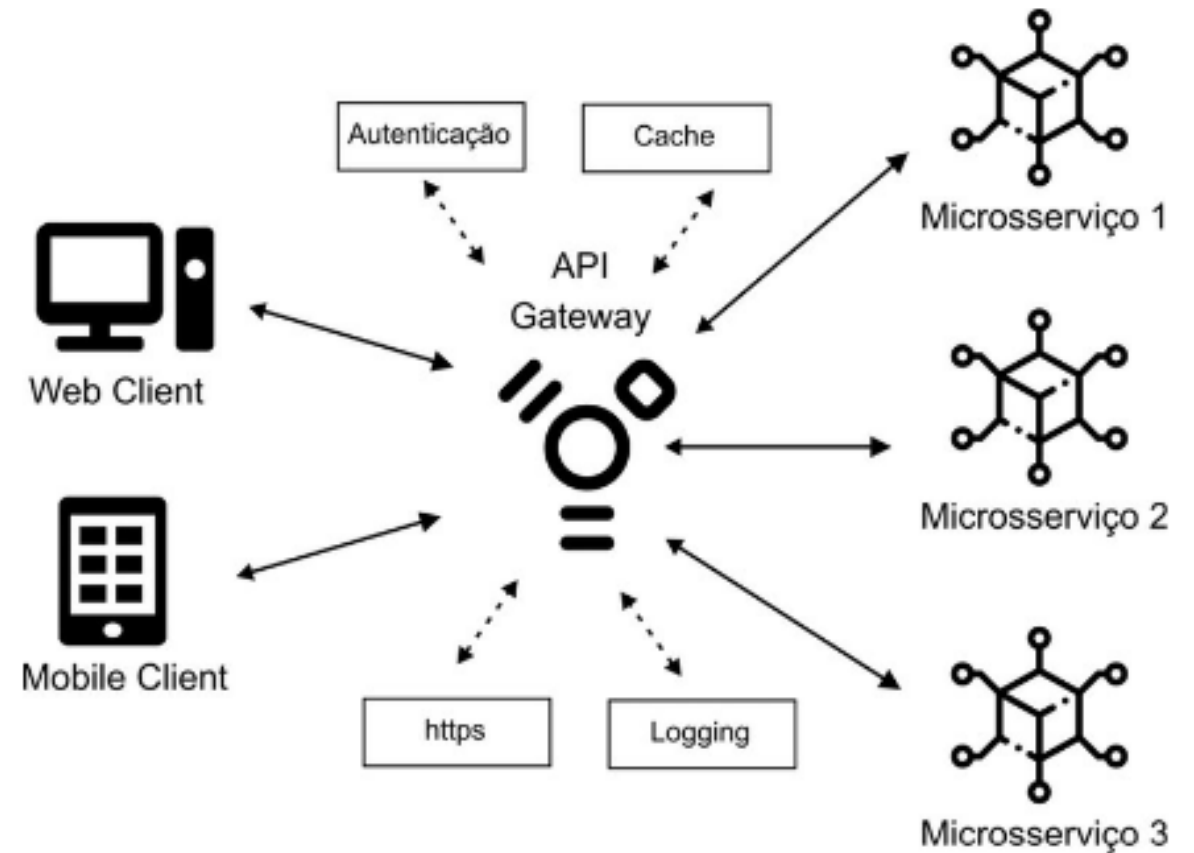
Microserviço sem nenhum mecanismo de centralização.

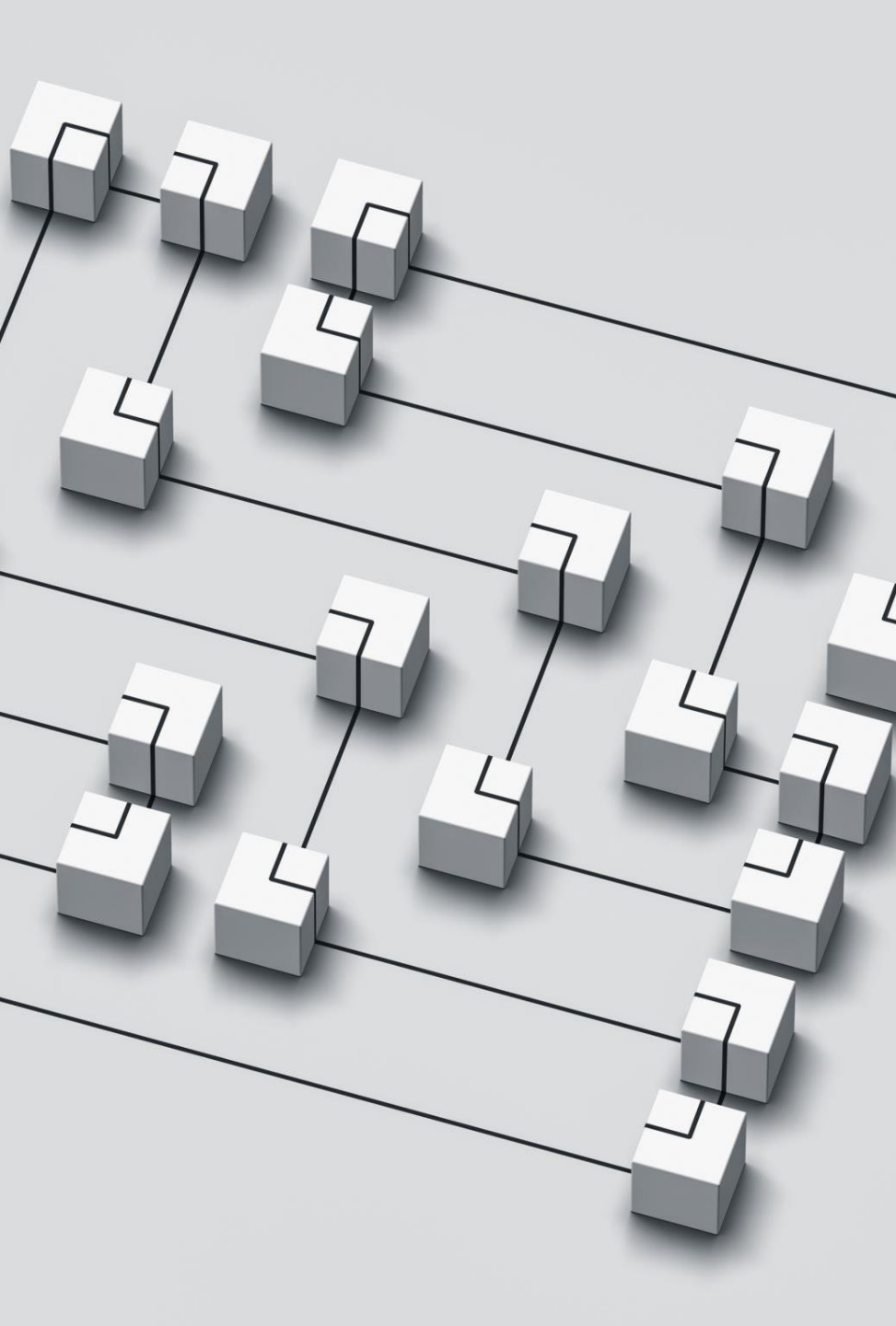
- Problemas
 - Segurança
 - Código duplicado
 - Log de Requisição e Métricas
 - Cache
 - Transformação e Orquestração
 - Roteamento
 - Cada cliente precisa tratar suas próprias rotas
 - Normalização de Protocolos



API Gateway

- Age como uma porta de entrada para acesso aos microsserviços.
- Pode ser implementado por código ou servidores dedicados.





Vantagens API Gateway

- É um ponto único de acesso onde são feitos o roteamento e a composição de requisições, a tradução de protocolo, logs e métricas das requisições.
- Eliminação de duplicidades, facilitando alterações e manutenção.
- Redução do tráfego de rede.
- Simplificação das chamadas dos clientes que não precisam conhecer as complexidades do backend.

Cuidados com a API Gateway

- **Ponto único de falha**
- Sugestão
 - Implantação com redundância
 - Escalabilidade Automática
 - Tolerância a falhas

