



Introdução

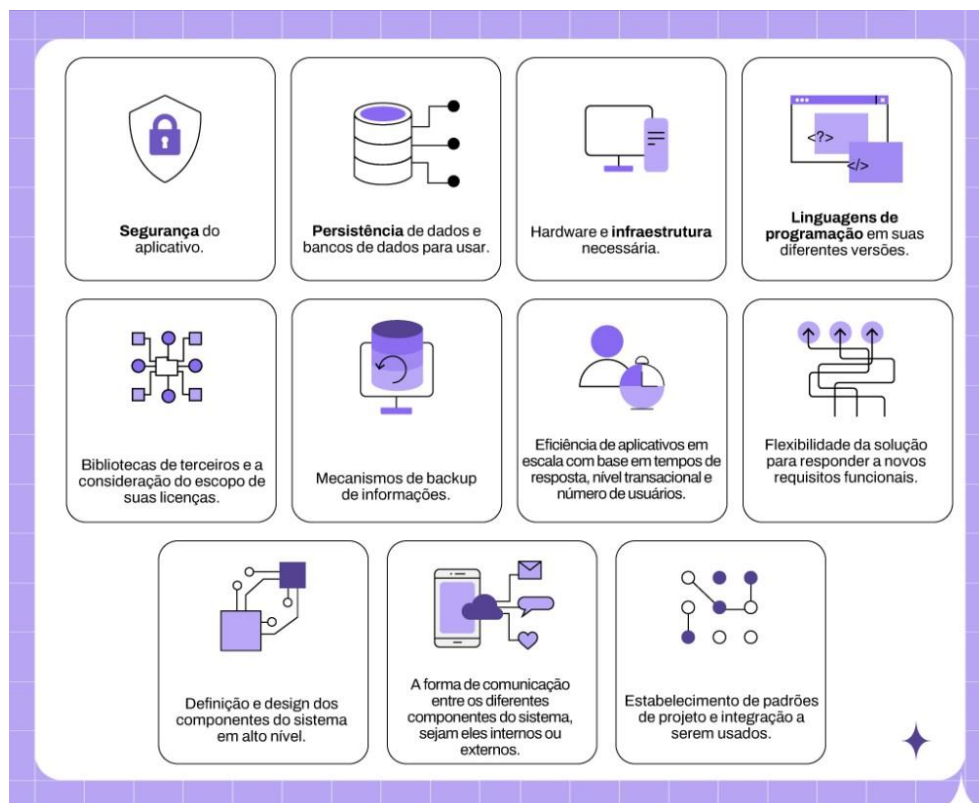
O que é arquitetura de software?

Antes de revisar sistemas monolíticos e as soluções orientadas a microsserviços, é importante que tenhamos identificado o conceito de arquitetura de software.

↙

Você já pensou em como é a base de uma **aplicação web** ou uma **API**?

Como usuários, vemos apenas uma parte de uma ótima solução que abrange diferentes aspectos. Vamos ver alguns deles abaixo.





Em suma, poderíamos dizer que a arquitetura de software estabelece as bases em termos de estrutura, operação e interação entre as partes de um sistema sobre o qual é construído um ecossistema de aplicações, além dos algoritmos utilizados.



Mas qual é a primeira coisa que nos vem à mente quando mencionamos a profissão de "arquiteto"? É provavelmente o arquiteto no setor da construção, ou seja, a pessoa que elabora um plano baseado nas exigências do futuro habitante da casa, considerando normas de construção para definir os espaços dos quartos, ventilação e luminosidade, portas de acesso, distribuição da canalização, características das fundações, suporte em peso das colunas e telhado de betão, paredes, etc. Este plano é a base que

mostrará aos trabalhadores, canalizadores, eletricitistas, instaladores a gás e pintores como realizar o trabalho.



Se fizermos uma analogia com o **arquiteto de software**, encontramos muitas semelhanças, não em questões técnicas, mas enquanto todas as definições tomadas (com base em normas) estão inter-relacionadas entre si, para um determinado fim conforme a necessidade de um usuário ou cliente.

Para que um arquiteto ponha em prática os seus conhecimentos técnicos e experiência profissional, precisa conhecer "o negócio" que enfrentará, ou seja, conhecer as necessidades dos usuários finais, o domínio empresarial, as regras a aplicar neste domínio e as características não-funcionais associadas, por exemplo: o número de usuários online concorrentes para um chat ou o número médio de pesquisas que existirão num e-commerce, disponibilidade da aplicação, etc.

Vale ressaltar que as fundações e a arquitetura definidas não são escritas em pedra e tendem a mudar com o tempo, mas é importante que este fator seja considerado pelo arquiteto de software para assegurar que a escalabilidade e as alterações ao ecossistema do software sejam efetuadas de forma simples e transparente sem afetar a disponibilidade do sistema.

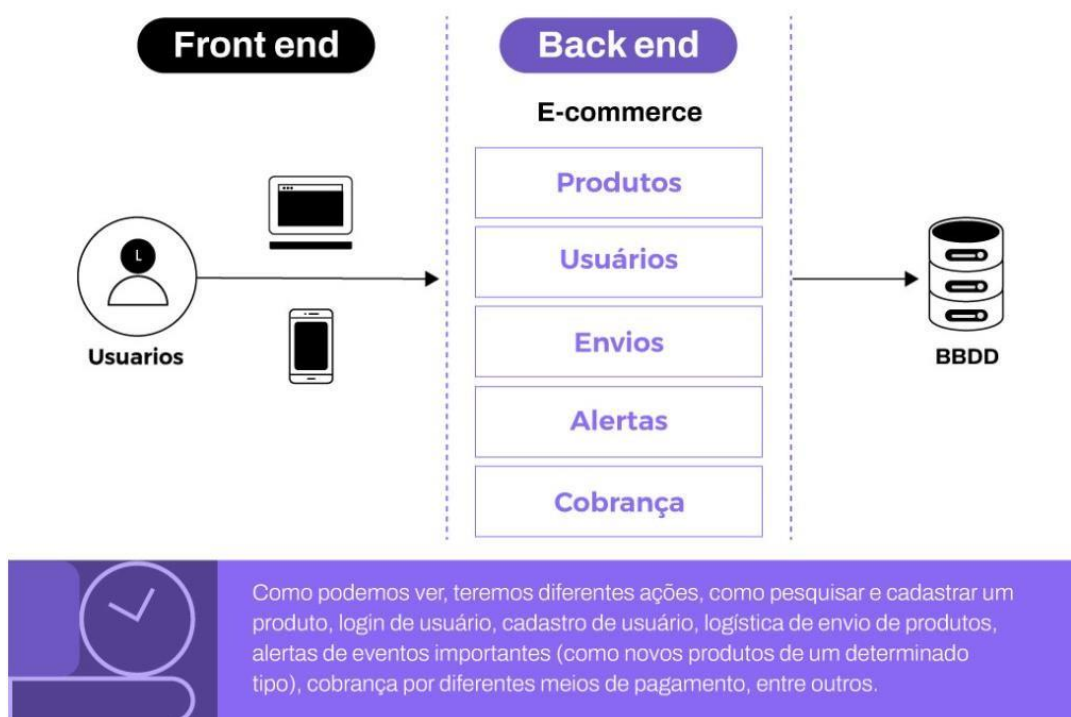


O que é arquitetura monolítica?

Agora que conhecemos o conceito de arquitetura de software, analisaremos a arquitetura monolítica. Pensaremos na evolução dos computadores, no início não havia conexão entre eles, ou seja, não havia Internet, e esses computadores ainda não eram pessoais, mas eram grandes centros de computação. Claramente, essa realidade tornou o primeiro e os programas subsequentes autossuficientes ou, em outras palavras, sistemas compostos por vários módulos inter-relacionados dentro de um mesmo computador.

Então, tanto o hardware quanto o software evoluíram de tal forma que os computadores pessoais e a inter-relação entre eles através de uma rede começaram a se desenvolver. Assim, não era mais necessário que os recursos estivessem localizados apenas no mesmo computador, e surgiram as primeiras páginas web de diferentes tipos de negócios, como home banking e e-commerce. Vejamos um exemplo deste último.

Se pensarmos em um e-commerce típico, podemos inferir diferentes módulos especificados no diagrama a seguir:





Podemos ver que absolutamente todas as funcionalidades estão integradas em uma única unidade de software, utilizando um único recurso de persistência de informações dentro de um banco de dados. Isso significa que, por exemplo, a lógica de negócios de produtos e a lógica de cobrança estarão juntas, com um acoplamento e coesão que dependerá da lógica definida pelo programador.

Agora veremos quais são as vantagens e desvantagens em diferentes situações da referida arquitetura, continuando com o mesmo exemplo.

Arquitetura monolítica

Vamos ver as **vantagens** e **desvantagens** dessa arquitetura em relação aos possíveis cenários.

- O que acontece se houver um erro?
- O acoplamento de soluções é um problema em termos de manutenção e implantação produtiva?
- O que acontece se eu quiser mudar a linguagem de programação para facilitar, por exemplo, a lógica de despacho?
- Do ponto de vista do usuário, o que aconteceria se nosso e-commerce sofresse um aumento inesperado de consumo devido a uma data especial como o Natal?
- O que acontece se nosso e-commerce começar a fazer sucesso e a demanda por novas funcionalidades dos módulos existentes crescer enormemente?

O que acontece se houver um erro?

Todos os sistemas apresentam erros em um ambiente produtivo. Tanto erros técnicos por falha de dependências e bancos de dados, como falhas funcionais —por exemplo, um erro no formato do e-mail—. Esses erros são facilmente detectáveis nesse tipo de arquitetura, pois a falha está em um único local que abrange todos os módulos ou componentes da solução.






? O acoplamento de soluções é um problema em termos de manutenção e implantação produtiva?

Embora o acoplamento de soluções possa ser um problema em termos de escalabilidade, não é em termos de manutenção e implantação produtiva. Manter um único repositório centralizado do negócio de comércio eletrônico é mais barato do que manter vários repositórios, desde que o número de desenvolvedores na equipe seja pequeno. Uma equipe muito grande causaria conflitos de código devido a alterações simultâneas. Além disso, **a implantação** é muito mais simples quando feita como um todo, e não como a soma das partes.


← →

 Vantagem

? O que acontece se eu quiser mudar a linguagem de programação para facilitar, por exemplo, a lógica de despacho?

Uma vez que o conjunto de linguagens de programação utilizadas para desenvolver a aplicação monolítica é único, visto que todos os componentes que respondem à aplicação trabalham em conjunto dentro de uma mesma unidade, devemos alterar a linguagem de toda a solução ou terceirizar alguns componentes.


← →

 Desvantagem

? Do ponto de vista do usuário, o que aconteceria se nosso e-commerce sofresse um aumento inesperado de consumo devido a uma data especial como o Natal?

O sistema ficaria saturado por não conseguir responder à procura agregada e teríamos tempos de resposta excessivos em todas as funcionalidades, o que se traduz em problemas de negócio, perda de vendas e lucros para a organização.

← →

 Desvantagem



? O que acontece se nosso e-commerce começar a fazer sucesso e a demanda por novas funcionalidades dos módulos existentes crescer enormemente?

Para atender às necessidades dos usuários e não perder para a concorrência, vamos exigir que a equipe cresça rapidamente em tamanho, a fim de reduzir o **TTM** (*time to market*, tempo que leva para colocar novos recursos em produção). A equipe atinge o tamanho esperado e todos começam a desenvolver editando o mesmo repositório (já que é um único sistema integrado). Assim, teremos conflitos de mudança de código como resultado, testes unitários falhando por ter dependência entre os módulos da solução. Isso indica que esta solução não é viável para sistemas que constantemente possuem atualizações e novos recursos.

Desvantagem

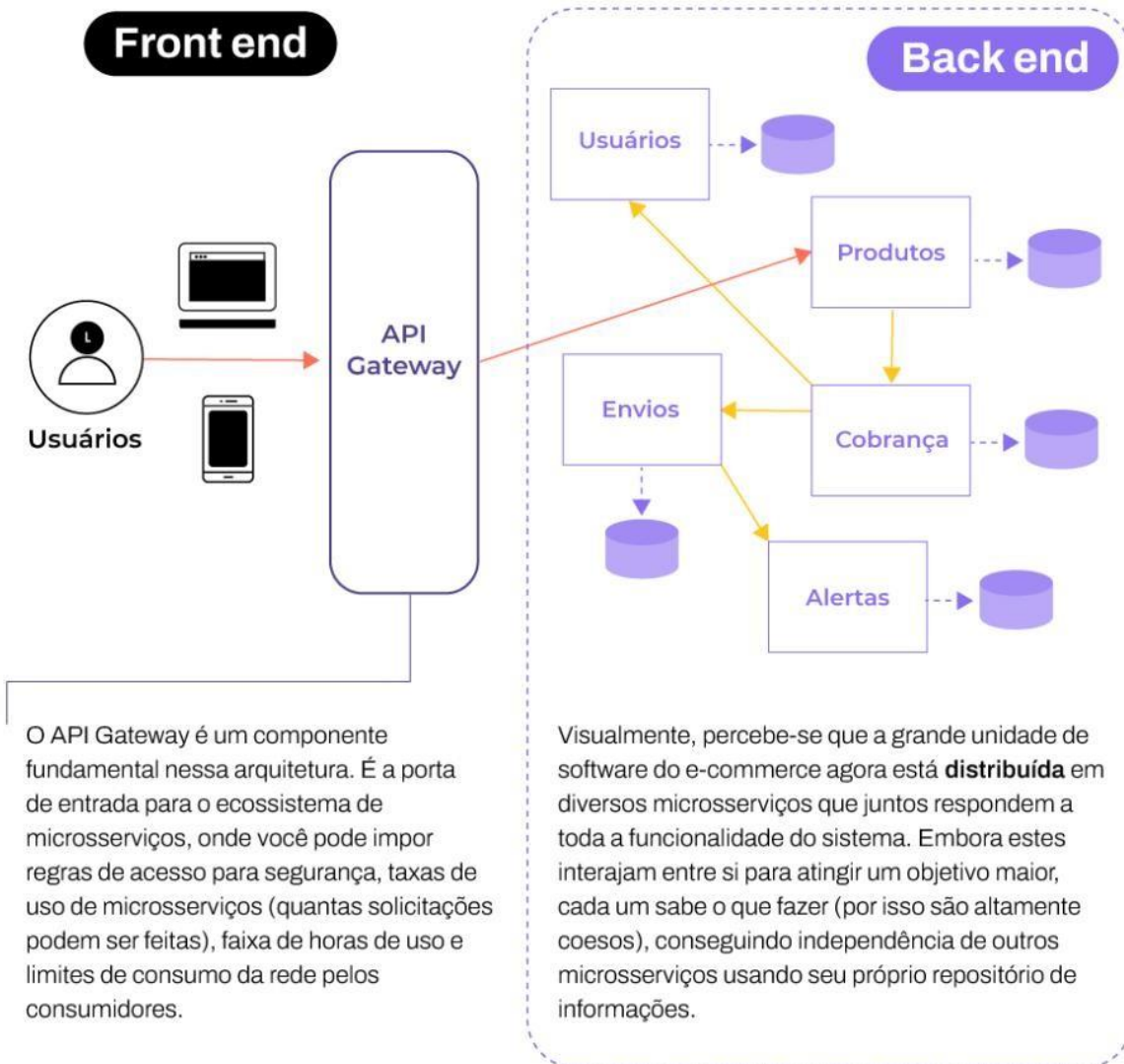
O que é a arquitetura dos microsserviços?

Esta arquitetura é a antítese da arquitetura monolítica, uma vez que ela não une as funcionalidades como um todo, mas cria pequenos componentes com uma única funcionalidade limitada. Estes têm autonomia em recursos computacionais e dependências como a BBDD, de modo que a sua evolução seja totalmente independente ao longo do tempo.

Um microsserviço não é necessariamente um serviço REST ou SOAP, mas é responsável pela prestação de um serviço particular — ou utilizando estes protocolos ou de alguma outra forma — como o processamento batch, reagindo a algum evento que ocorre através de mensagens em fila de espera, processamento de algum arquivo via FTP, etc.



Agora apresentamos nossa aplicação monolítica inicial de e-commerce baseada na arquitetura de microsserviços.



Finalmente, veremos as vantagens e desvantagens desta arquitetura em diferentes situações possíveis.



Arquitetura de microsserviços

Vamos ver as **vantagens** e **desvantagens** dessa arquitetura em relação aos possíveis cenários.

- O que aconteceria se o consumo do aplicativo crescesse no Natal, as buscas aumentassem por causa das promoções e o site começasse a ficar saturado?
- O que acontece com o TMM (tempo mínimo para que o produto ou funcionalidade seja produtivo)?
- O que acontece se eu quiser mudar a linguagem de programação?
- Que benefício há agora que um grande sistema é a soma de suas partes?
- O que acontece se houver uma falha?
- Por que a depuração e a descoberta de erros são mais complexas?
- Por que o desempenho pode ser afetado?



O que aconteceria se o consumo do aplicativo crescesse no Natal, as buscas aumentassem por causa das promoções e o site começasse a ficar saturado?

Nessa arquitetura, precisaríamos apenas aumentar o número de instâncias de microsserviços do produto e não toda a solução! Desta forma, podemos atacar o problema sem desperdiçar recursos. Recursos que serão reduzidos após a temporada de compras de Natal. Isso é possível devido à independência da infraestrutura e ao desacoplamento da solução. Este é o que chamamos **escalabilidade**.



Vantagem



O que acontece com o TMM (tempo mínimo para que o produto ou funcionalidade seja produtivo)?

Isso é aprimorado e otimizado nesse tipo de arquitetura, pois uma grande equipe de desenvolvedores terá total **agilidade** para a evolução de cada microsserviço com seu próprio repositório. Por exemplo, seremos capazes de responder de forma otimizada à necessidade de novas funcionalidades de negócios estabelecidas pela gestão de envios sem afetar todo o sistema.



Vantagem



O que acontece se eu quiser mudar a linguagem de programação?

Sendo uma arquitetura distribuída com componentes totalmente independentes, podemos ser políglotas quanto à linguagem de programação. Ou seja, ter microsserviços desenvolvidos em Java, outros em Microsoft .NET, Node.js, etc. De acordo com a natureza da funcionalidade a ser satisfeita, a linguagem mais adequada é escolhida.



Vantagem



Que benefício há agora que um grande sistema é a soma de suas partes?

Podemos dizer que aquelas partes com um único propósito são totalmente **reutilizáveis** no ecossistema da organização. Vamos pensar no módulo de cobrança do nosso e-commerce, se a organização crescer e —além do e-commerce— houver outros sites de venda ou cobrança de serviços, esse microsserviço de cobrança teria novos sistemas de consumo.



Vantagem



O que acontece se houver uma falha?

As falhas neste tipo de arquitetura são mais complicadas de descobrir e resolver. Como um microsserviço chama outro para atingir um objetivo de nível superior em um sistema padrão, cada chamada pode ser uma falha que deve ser controlada para não perder a integridade do nosso sistema e responder adequadamente, sendo tolerante a falhas.



Desvantagem



Por que a depuração e a descoberta de erros são mais complexas?

Cada microserviço tem seu log e contexto de execução, portanto, o rastreamento de chamadas entre microserviços deve ser realizado de forma obrigatória para identificar a cadeia de chamadas de uma solicitação do usuário e estabelecer a correção da falha correspondente.

Desvantagem

Por que o desempenho pode ser afetado?

Cada microserviço tem uma latência de rede quando é invocado. O tempo de resposta é a soma da cadeia de microserviços invocados, portanto, é preciso ter cuidado com o uso de recursos, algoritmos implementados e tamanhos de resposta para garantir um tempo de resposta ideal.

Desvantagem