Microsserviços e DHT

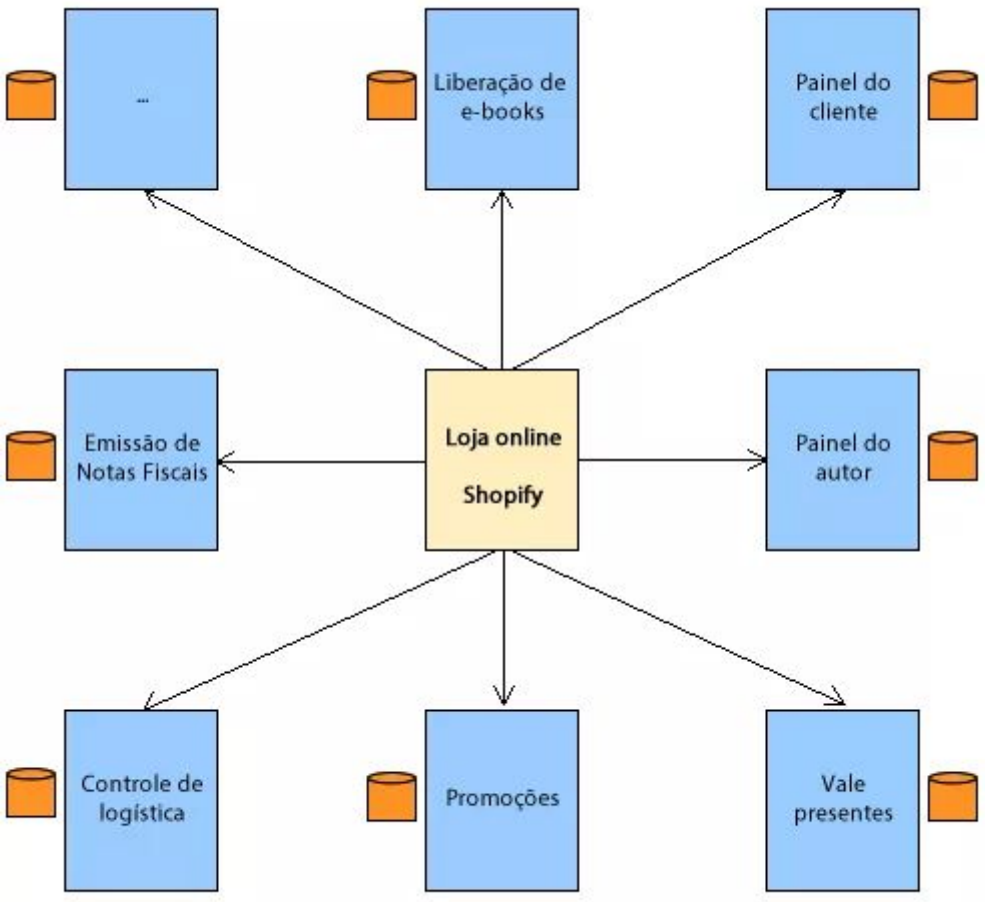
Microsserviços vs Aplicações monolíticas

São vários “sistemas” com funções específicas, como autenticação, edição de perfil, cadastro de conteúdo, alterações, exclusões e outros. Estes são isolados um do outro, não se conhecendo, porém integrando uma aplicação comum. A aplicação pode ser denominada produto, ao qual faz as chamadas dos microsserviços individualmente, de acordo com a necessidade.

Microsserviços devem sempre ser independentes e representam o contexto de negócio de uma aplicação, cálculo de frete, por exemplo. A vantagem é que são independentes e múltiplos, leves e têm seu próprio modelo de dados exclusivo. Os protocolos comuns de mensagens usados são HTTP e REST. A facilidade de manutenção sem prejudicar a operação de toda a aplicação é um dos destaques, bem como problemas de disponibilidade.

Ao exemplo de uma loja online, é possível ter vários microsserviços. A loja seria o produto/aplicação comum entre todos.

Microsserviços: fechar pedido, criar conta, alterar dados da conta, calcular frete, etc.



Fonte: http://blog.caelum.com.br/arquitetura-de-microservicos-ou-monolitica/

No caso de aplicações monolíticas não há de forma clara a independência destes serviços. O modelo de dados é interdependente, bem como a chamada de funções. Desta forma torna-se mais complexo tratar questões como disponibilidade, manutenção e prevenção de falhas.

Comparativo

Tipo	Microsserviços	Aplicações Monolíticas
Modelo de dados	Individual de cada serviço, alinhado ao resultado esperado	Alinhado aos padrões da aplicação comum
Denominação na programação	Módulos/serviços	Funções
Independência de código	✔	❌

Histórico

Segundo Newman, microsserviços são serviços com poucas responsabilidades, pequenos e autônomos que podem trabalhar de forma independente ou em conjunto com outros serviços. Ao contrário de sistemas monolíticos, cada micro serviço é empacotado em um artefato separado, e portanto pode-se realizar a implantação de cada um independentemente.

Prós e contras

Vantagens

- Heterogeneidade Tecnológica
  Pode-se escolher a melhor linguagem/ferramenta para a execução de cada tarefa (microsserviço).
  Exemplo: em uma rede social, um microsserviço para armazenar as relações entre os usuários em um banco de dados orientado a grafos, e outro microsserviço para armazenar os posts em um banco de dados orientado a documentos.
- Resiliência
  É a habilidade dos sistemas de recuperação de falhas. No monolítico, se uma parte falha, o sistema todo falha. Com microsserviços, caso um falhe é possível isolar o problema.
  Exemplo: Netflix, se o serviço de recomendações falhar, o serviço de streaming continua operando.
- Escalabilidade

Em aplicações monolíticas escala tudo ou nada. Com microsserviços pode-se escolher quais partes precisam ser atualizadas.

- Facilidade de Implantação
  É possível realizar a implantação somente da feature que será lançada. Em aplicações monolíticas as implantações são longas, demoradas e geralmente são realizadas poucas entregas com muitas features e correções, o que requer muitos testes.

Desvantagens

- Complexidade de Desenvolvimento
  Em sistemas monolíticos, escreve-se tudo em um único projeto, realizando chamadas internas entre classes onde o único requisito para tal é a importação de uma classe em outra.
  Em aplicações orientadas a serviços, é necessário gerenciar as dependências entre eles. Mesmo os microsserviços sendo escritos para trabalharem de maneira autônoma, com ou sem as partes com as quais eles interagem, se os diferentes módulos do projeto não forem atualizados os microsserviços não tirarão proveito das novas features que forem lançadas.
- Chamadas Remotas
  Chamadas remotas são muito mais custosas do que chamadas a classes internas. Se os serviços forem pequenos demais a comunicação via rede se tornará um problema. O ideal é que o microsserviço seja pequeno o suficiente para executar sua funcionalidade, sem necessidade de processos adicionais que não fazem parte de seu propósito.
- Gerenciamento de múltiplos bancos de dados e transações
  É necessário um grande esforço inicial para realizar todas as configurações necessárias e ainda têm alto custo para manter vários bancos de dados funcionando. Os controles transacionais em sistemas distribuídos são extremamente complexos, uma vez que precisa ser realizada uma série de tratamentos em caso de falha.

DHT

Segundo a Wikipedia, DHTs são Tabelas hash distribuídas ou ainda tabelas de espalhamento distribuídas são uma classe de sistemas distribuídos descentralizados que provêem um serviço de lookup similar a uma tabela hash: pares (chave, valor) são armazenados na DHT e qualquer nó participante pode eficientemente recuperar o valor associado a uma dada chave. A responsabilidade de manter o mapeamento de chaves para valores é distribuída entre os nós tal que mudanças no conjunto de participantes causem o mínimo de desordem. Isso faz com que as DHTs escalem a um número extremamente grande de nós e gerencie chegadas, saídas e falhas contínuas dos mesmos. <Wikipedia, https://pt.wikipedia.org/wiki/Distributed_hash_table>

Os DHTs usam um variante de hashing consistente (no qual uma pequena alteração no nome não altera significativamente o hash) que utiliza uma função f(k1, k2) que define uma distância abstrata entre k1 e k2, que não tem a ver com a distância geográfica nem

com o tempo de latência. Cada nó recebe um identificador (ID). Se um nó recebe um ID x, ele recebe todas as chaves km para as quais x é o ID mais próximo (por exemplo, o Chord, um dos protocolos DHT mais usados, usa um círculo no sentido horário), que é calculado por f(km,x).

A escolha de quem possuirá k varia, mas todos tem um princípio básico: será escolhido um nó que tenha k em seu ID ou que tenha ligação com um nó que tenha o ID mais parecido com k. è utilizado, então, um algoritmo para encontrar esse nó. Esse tipo de roteamento é chamado key-based routing.

Histórico

Ele foi criado para reunir as vantagens dos sistemas de buscas do Napster e da Gnutella. No Napster, ao entrar na rede, cada nó enviava ao servidor central uma lista dos arquivos que possuía. O servidor, então se encarregava de fazer as buscas no seu banco de dados. Era muito eficiente, mas tornava o Napster vulnerável à ataques judiciais, pois permitia provar que a rede guardava e direcionava arquivos ilegais. A Gnutella usava um modelo de inundação no qual, quando a busca era feita, todas as máquinas conectadas à rede recebiam uma mensagem solicitando o determinado arquivo. Esse método era muito menos eficiente, pois tornava a comunicação lenta. Era preciso criar um sistema que tivesse a mesma eficiência do Napster, mas que fosse descentralizado. Um problema do DHT é que ele só realiza buscas pelo nome exato do arquivo ao invés de usar palavras-chave.