2021/04/19 - AULA 03.1

Laboratório de Sistemas Computacionais Complexos

https://uclab.xyz/sistemas-complexos-2021-aula03-1

Renato Cordeiro Ferreira
renatocf@ime.usp.br

João Francisco Daniel
joaofran@ime.usp.br

Alfredo Goldman
gold@ime.usp.br

Thatiane de Oliveira Rosa
thatiane@ime.usp.br

Em caso de dúvidas

Acessem slido.com com #complexos

ou

Agenda

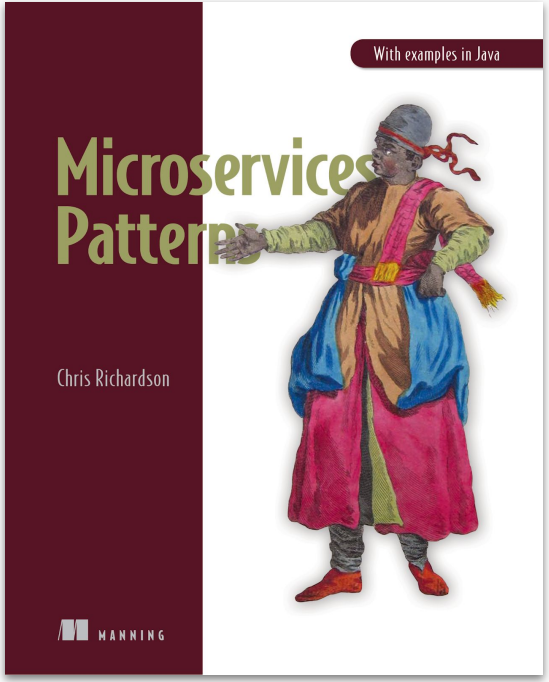
Tema da aula:
Introdução aos Padrões de Microsserviços

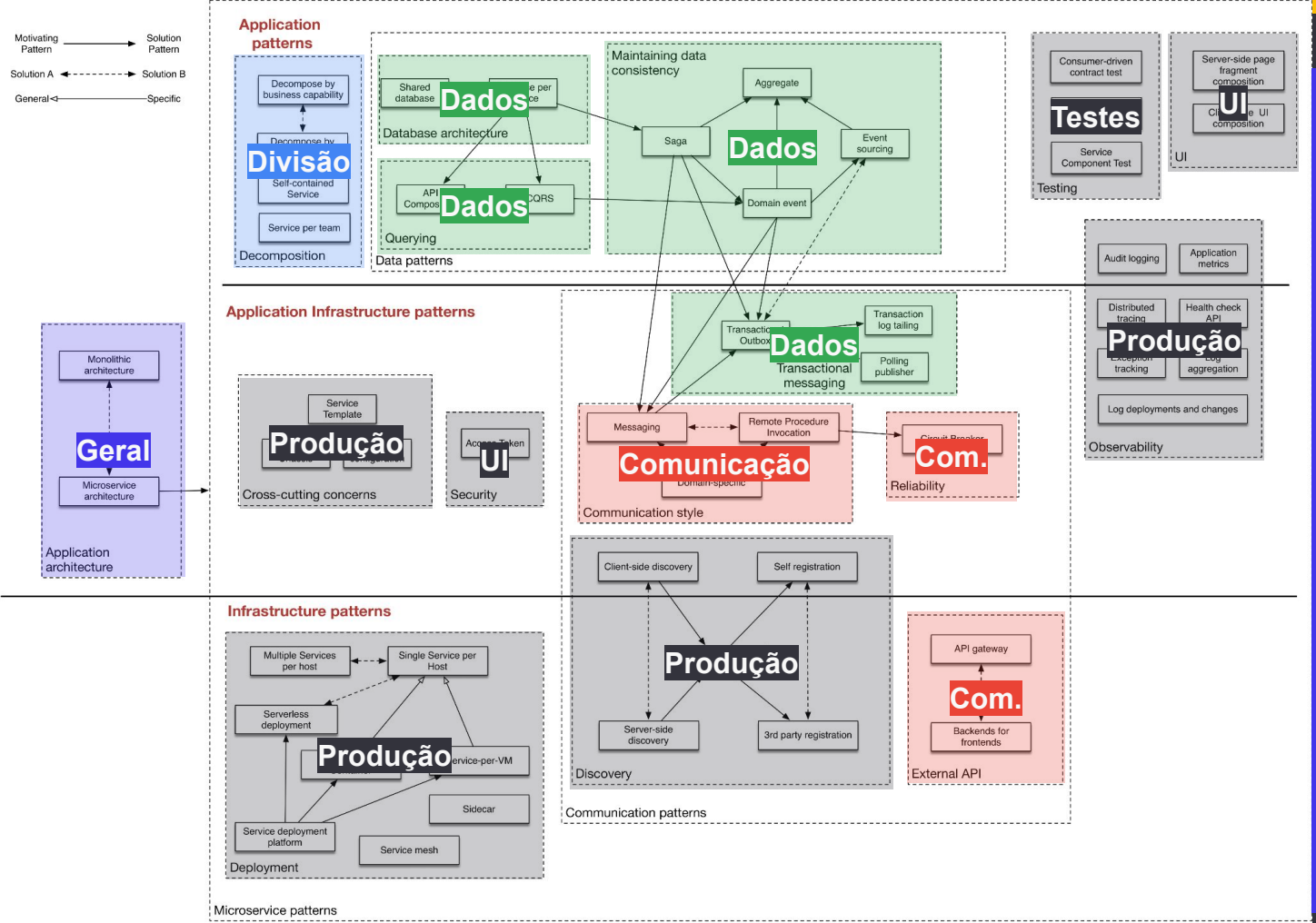
1. O que é um padrão?
2. Linguagem de Padrões de Microsserviços
3. Monolito vs. Microsserviços
4. Desafios do uso de Microsserviços
5. Divisão Serviços: Lei de Conway + Serviço por Time
6. Comunicação Interna: Remote Procedure Invocation vs. Messaging
7. Comunicação Externa: API Gateway vs. Back-End for Front-End

O que é um padrão?

*"Um padrão é uma solução reutilizável para um problema que ocorre em um contexto particular. É uma ideia que tem suas origens na arquitetura do mundo-real e tem se provado útil para projeto e arquitetura de software. O conceito de padrão foi criado por **Christopher Alexander**, um arquiteto. Ele também criou o conceito de linguagem de padrão, uma coleção de padrões relacionados que resolvem problemas dentro de um domínio em particular."*

-- Chris Richardson, **Microservices Patterns** (Capítulo 1)





Microservices Pattern Language

Chris Richardson

Monolito vs. Microsserviços

Problema

Qual a arquitetura de deploy da aplicação?

Forças

- Existe um time de desenvolvedores trabalhando na aplicação
- Novos membros do time devem se tornar produtivos rapidamente
- A aplicação deve ser fácil de entender e modificar
- Você quer praticar a entrega contínua da aplicação
- Você deve executar múltiplas instâncias da aplicação em múltiplas máquinas para satisfazer os requisitos de disponibilidade e escalabilidade
- Você quer tirar vantagem de tecnologias emergentes (frameworks, linguagens de programação, etc.)

Monolito vs. Microsserviços

Monolito

Estruture a arquitetura da aplicação como um único programa empacotado como executável

- Mais **simples** de desenvolver
  (único ambiente requer poucas IDEs, ferramentas, etc.)
- Mais **simples** de entregar
  (única aplicação executável para deploy)
- Mais **simples** de escalar
  (criação de mais cópias da única aplicação)
- Mais **complexo** de entender
  (única aplicação contém todo código conjunto)
- Mais **complexo** de adotar tecnologias
  (única aplicação executada sobre um ambiente)

Microsserviços

Estruture a arquitetura da aplicação como um conjunto de serviços colaborativos fracamente acoplados

- Mais **complexo** de desenvolver
  (múltiplos ambientes requer várias IDEs, ferramentas, etc.)
- Mais **complexo** de entregar
  (múltiplas aplicações executáveis para deploy)
- Mais **complexo** de escalar
  (criação de mais cópias de múltiplas aplicações)
- Mais **simples** de entender
  (cada aplicação contém código de um contexto)
- Mais **simples** de adotar tecnologias
  (cada aplicação executada sobre um ambiente)

Monolito vs. Microsserviços

Monolito

Estruture a arquitetura da aplicação como um único programa empacotado como executável

- Mais **restrito** de desenvolver
  (único ambiente requer poucas IDEs, ferramentas, etc.)
- Mais **restrito** de entregar
  (única aplicação executável para deploy)
- Mais **restrito** de escalar
  (criação de mais cópias da única aplicação)
- Mais **restrito** de entender
  (única aplicação contém todo código conjunto)
- Mais **restrito** de adotar tecnologias
  (única aplicação executada sobre um ambiente)

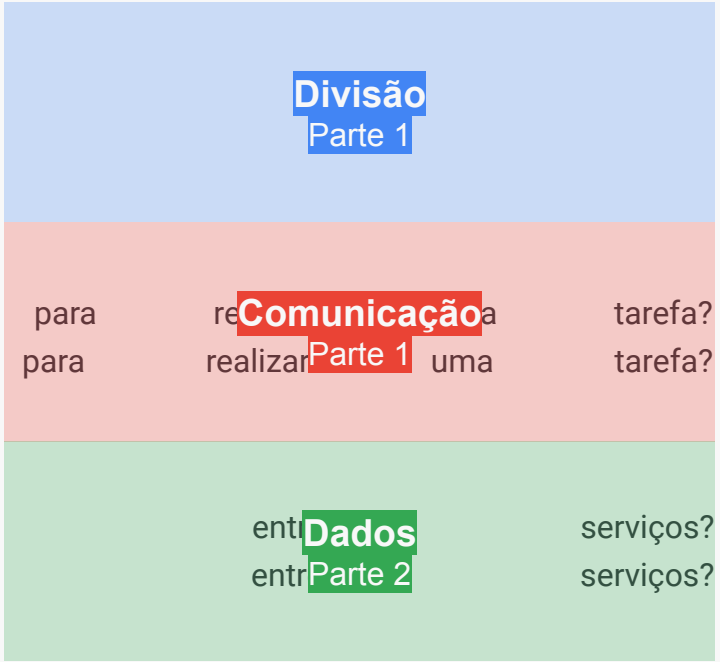
Microsserviços

Estruture a arquitetura da aplicação como um conjunto de serviços colaborativos fracamente acoplados

- Mais **flexível** de desenvolver
  (múltiplos ambientes requer várias IDEs, ferramentas, etc.)
- Mais **flexível** de entregar
  (múltiplas aplicações executáveis para deploy)
- Mais **flexível** de escalar
  (criação de mais cópias de múltiplas aplicações)
- Mais **flexível** de entender
  (cada aplicação contém código de um contexto)
- Mais **flexível** de adotar tecnologias
  (cada aplicação executada sobre um ambiente)

Desafios do uso de Microsserviços

1. **Divisão**
   - Como definir a responsabilidade de cada serviço?
   - Como definir a quantidade de serviços?
   - Como definir o tamanho dos serviços?
2. **Comunicação**
   - Como coordenar múltiplos serviços para realizar uma tarefa?
   - Como acessar múltiplos serviços para realizar uma tarefa?
   - Como resistir à perda de um serviço?
3. Dados
   - Como dividir dados entre serviços?
   - Como juntar dados entre serviços?
   - Como replicar dados entre serviços?



Divisão de Serviços

agente	requisito
autonomia	complexidade

Problema

Qual o relacionamento entre times e serviços?

Forças

- Um time deve ser pequeno, e.g., 5-9 pessoas
- Um time deve ser autônomo e fracamente acoplado
- O tamanho e complexidade da base de código não deve exceder a capacidade cognitiva do time
- Decomposição mais granular dos serviços melhora as -idades (requisitos não-funcionais) incluindo manutenibilidade, testabilidade e entregabilidade
- Decomposição mais granular dos serviços adiciona complexidade

Divisão de Serviços

agente	requisito
autonomia	complexidade

"Qualquer organização que projeta um sistema (definido amplamente) irá produzir um design cuja estrutura é uma cópia da estrutura de comunicação da organização."

-- Melvin E. Conway, 1968

Serviço por Time

Cada serviço pertence a um **único time**, que tem a **responsabilidade exclusiva de alterá-lo**.

- Habilita autonomia com coordenação mínima entre times
- Habilita fraco acoplamento entre times
- Melhora qualidade do código pela posse de longo prazo
- Permite desalinhamento de times com implementação de funcionalidades voltadas para o usuário final
- Requer colaboração entre times para implementação de funcionalidades que incluem múltiplos serviços

Comunicação Interna

agente	requisito
autonomia	complexidade

Problema

Como serviços numa arquitetura de microsserviços se comunicam?

Forças

- Serviços com frequência precisam se comunicar.
- Comunicação síncrona resulta em acoplamento forte, ambos cliente e servidor precisam estar disponíveis pela duração da requisição.

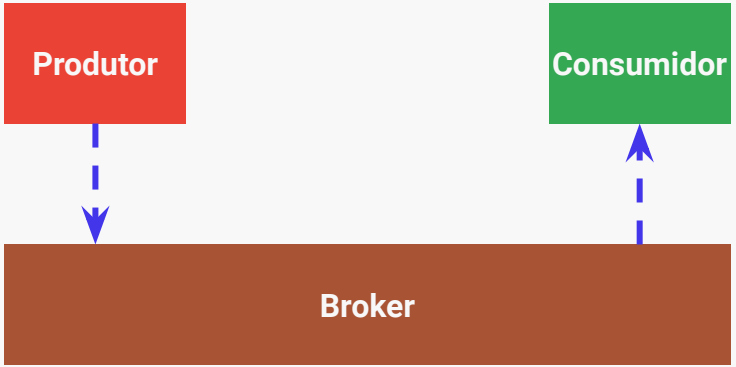
Comunicação Interna

Remote Procedure Invocation

Use **invocação de procedimento remota** para comunicação entre serviços. Os serviços se comunicam via **protocolo requisição-resposta**.

Messaging

Use **troca de mensagens assíncrona** para comunicação entre serviços. Os serviços se comunicam via **canais de troca de mensagens**.



Comunicação Interna

agente	requisito
autonomia	complexidade

Remote Procedure Invocation

Use **invocação de procedimento remota** para comunicação entre serviços. Os serviços se comunicam via **protocolo requisição-resposta**.

- Aumenta acoplamento durante execução pois atrela funcionamento de cliente e servidor da requisição.
- Requer utilização de protocolos de rede que geralmente não demandam infraestrutura adicional.
- Possui disponibilidade reduzida pois protocolo de rede exige que cliente e servidor estejam disponíveis durante a interação requisição-resposta.

Messaging

Use **troca de mensagens assíncrona** para comunicação entre serviços. Os serviços se comunicam via **canais de troca de mensagens**.

- Diminui acoplamento durante execução pois desatrela funcionamento de produtor e consumidor da mensagem.
- Requer utilização de um sistema de mensageria que precisa ser altamente disponível.
- Possui alta disponibilidade pois sistema de mensageria armazena mensagens do produtor até que o consumidor possa processá-las.

Comunicação Externa

agente	requisito
autonomia	complexidade

Problema

Como clientes de uma aplicação baseada em microsserviços acessa serviços individuais?

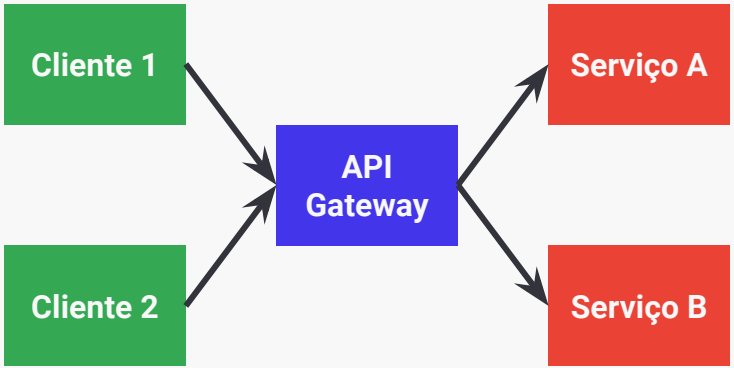
Forças

- A granularidade das APIs servidas por serviços costuma ser diferente do que clientes precisam: serviços tipicamente provêm APIs mais especializadas, o que significa que clientes precisam interagir com múltiplos serviços para obter a resposta desejada.
- Diferentes clientes possuem requisitos de dados diferentes, e.g., aplicativo mostra uma versão simplificada de uma página do navegador por conta do tamanho do dispositivo.
- Diferentes clientes possuem performance de rede diferentes, e.g., celular conectado à uma rede móvel possui menos banda que um desktop conectado à uma rede banda larga.
- Número de instâncias e localização (endereço + porta) dos serviços mudam dinamicamente.
- Particionamento entre serviços pode mudar com o tempo e deveria ser escondido dos clientes.
- Serviços deveriam usar uma diversidade de protocolos, alguns dos quais deveriam ser amigáveis à Web.

Comunicação Externa

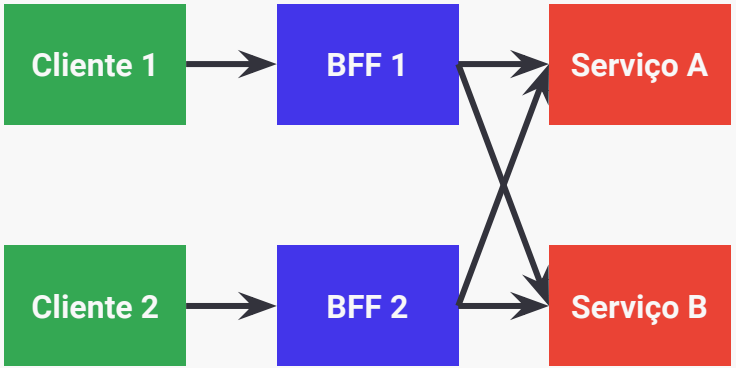
API Gateway

Implemente uma **API Gateway** como o único ponto de entrada para **todos os clientes**, que redireciona as requisições para um ou mais serviços.



Back-End for Front-End

Implemente um **Back-End for Front-End** como ponto de entrada para **cliente específico**, que redireciona as requisições para um ou mais serviços.



Comunicação Externa

agente	requisito
autonomia	complexidade

API Gateway

Implemente uma **API Gateway** como o único ponto de entrada para **todos os clientes**, que redireciona as requisições para um ou mais serviços.

- Isola clientes de conhecer a divisão em serviços.
- Isola clientes de determinar a localização dos serviços.
- Provê uma API única para todos clientes.
- Reduz o número de interações entre cliente e servidor.
- Requer manter serviço extra para servir clientes.
- Aumenta tempo de resposta por camada extra de rede.
- Permite implementação de funcionalidades transversais.

BFF (Back-End for Front-End)

Implemente um **Back-End for Front-End** como ponto de entrada para **cliente específico**, que redireciona as requisições para um ou mais serviços.

- Isola clientes de conhecer a divisão em serviços.
- Isola clientes de determinar a localização dos serviços.
- Provê uma API especializada para cada cliente.
- Reduz o número de interações entre cliente e servidor.
- Requer manter serviço extra para servir cada cliente.
- Aumenta tempo de resposta por camada extra de rede.
- Permite implementação de funcionalidades transversais.

Licença

Estes slides são concedidos sob uma Licença Creative Commons. Sob as seguintes condições: **Atribuição, Uso Não-Comercial e Compartilhamento pela mesma Licença**

Mais detalhes sobre essa licença em: creativecommons.org/licenses/by-nc-sa/3.0/

Créditos

Imagens usadas nesta apresentação são provenientes de: freepik.com

Frequência

Senha do Estudante: **xp7j9h**

2021/04/19 - AULA 03.1

Laboratório de Sistemas Computacionais Complexos

https://uclab.xyz/sistemas-complexos-2021-aula03-1

Renato Cordeiro Ferreira
renatocf@ime.usp.br

João Francisco Daniel
joaofran@ime.usp.br

Alfredo Goldman
gold@ime.usp.br

Thatiane de Oliveira Rosa
thatiane@ime.usp.br