Microsserviços: Ontem, Hoje e Amanhã

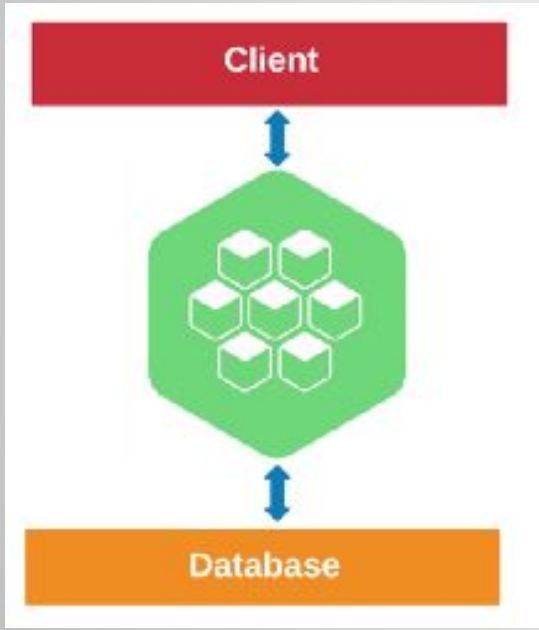
Nicola Dragoni, Saverio Giallorenzo, Alberto Lluch Lafuente,
Manuel Mazzara, Fabrizio Montesi, Ruslan Mustafin e Larisa Safina

Arthur Reznik Martins	2016001460
Caio Noriyuki Sasaki	2016009842
Guilherme Delgado de Carvalho da Costa Braga	2019003806
Ivan Leoni Vilas Boas	2018009073
Karen de Souza Pompeu	2016001610
Rodrigo Filippo Dias	2016001479

Professor: Rafael de Magalhães Dias Frinhani
Disciplina: Sistemas Distribuídos
Universidade Federal de Itajubá
Outubro de 2020

Introdução

Arquitetura Monolítica



Aplicações Monolíticas:

- Módulos formam uma estrutura única
- Módulos são dependentes um do outro
- Módulos compartilham mesmos recursos

Introdução

Problemas das grandes Aplicações Monolíticas:

1- Alta Complexidade no código (difícil evolução)

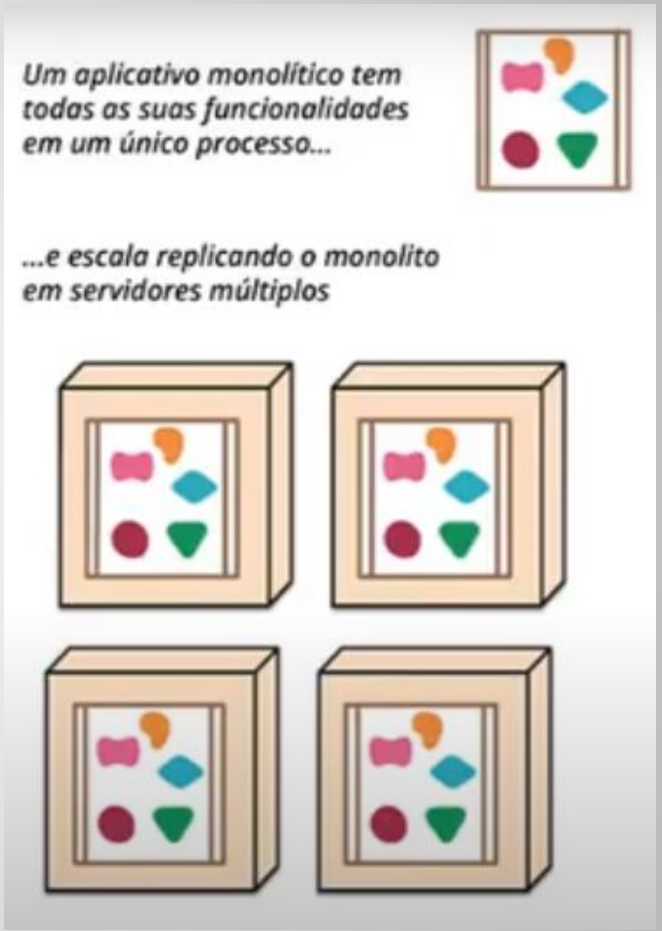
2- Alta dependências de componentes do código

3- Indisponibilidade

4- Tecnologia Estática

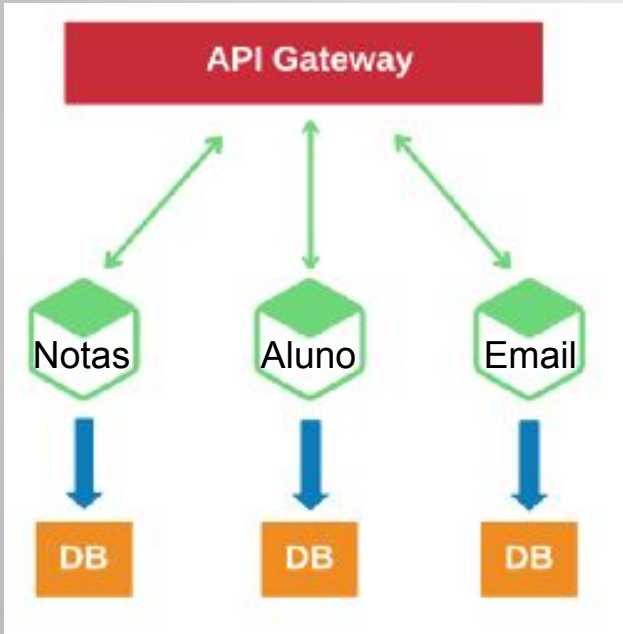
5- Difícil escalabilidade

6- Linguagem única



Introdução

Arquitetura de Microsserviços



Aplicações de Microsserviços:

- Módulos são independentes
- Módulos são coesos
- Módulos com banco de dados dedicado

Introdução

Soluções fornecidas pelos Microsserviços:

1- Baixa Complexidade no código
    1.1.Fácil manutenção e evolução

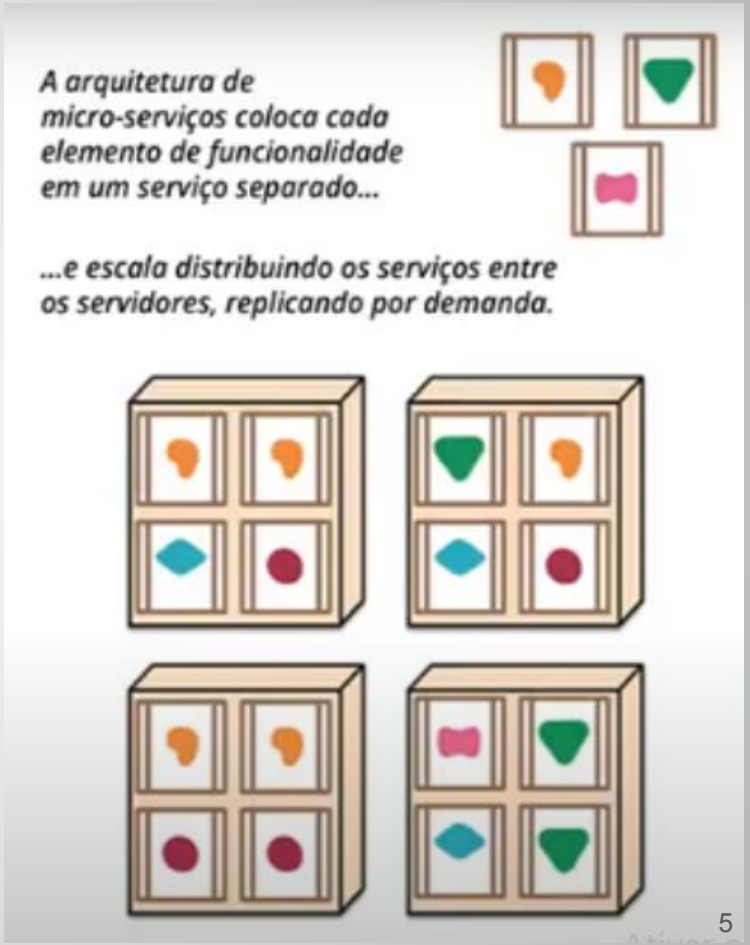
2- Sem dependências de componentes do código
    2.1 - Atualizações independentes e rápidas

3- Alta Disponibilidade

4- Tecnologia Flexível

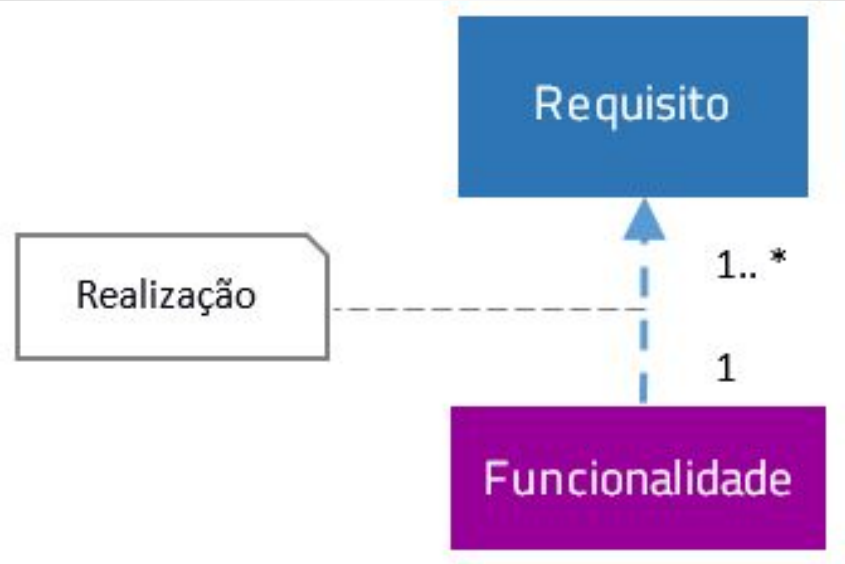
5- Alta escalabilidade

6- Linguagens múltiplas



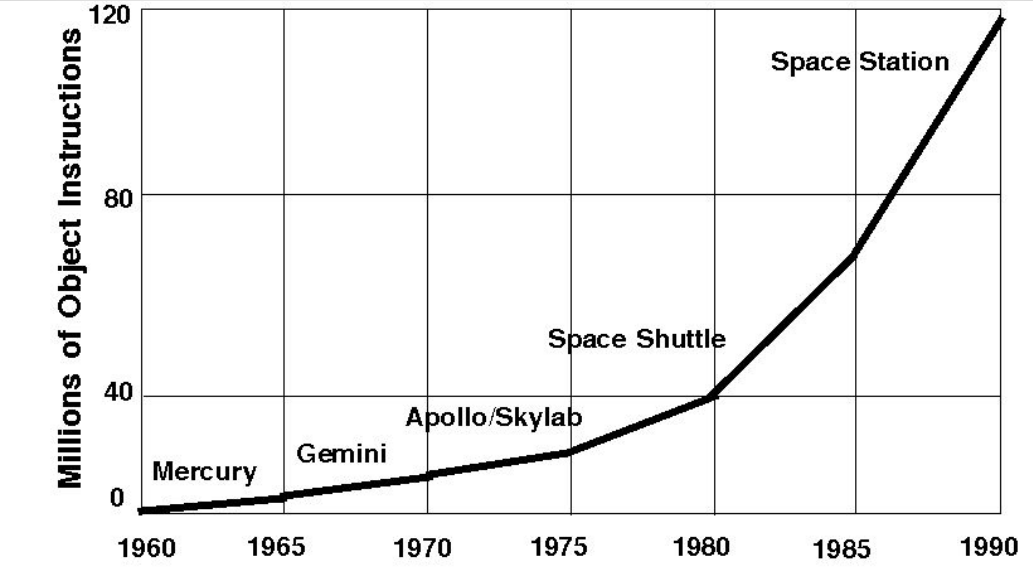
Ontem

Arquitetura é o que permite com que os sistemas se desenvolvam e forneçam um nível de serviço ao longo do seu ciclo de vida



Ontem
Dos primeiros dias ao padrões OO

Anos 60 - Problemas no desenvolvimento de softwares de larga-escala

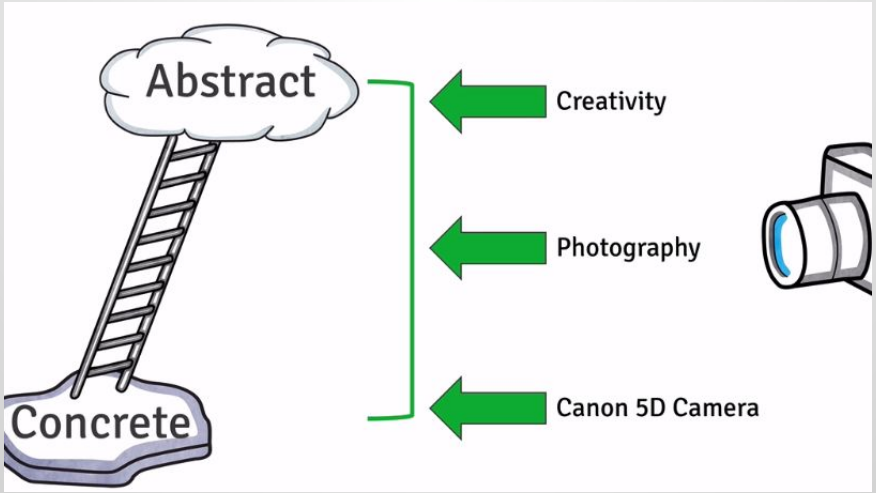


Complexidade do tamanho do Software. Fonte: the Space Effort

Ontem
Dos primeiros dias ao padrões OO

Anos 60 - Problemas no desenvolvimento de softwares de larga-escala

Anos 70 - Design de software e implicações no desenvolvimento



Design de Software.

Ontem
Dos primeiros dias ao padrões OO

Anos 60 - Problemas no desenvolvimento de softwares de larga-escala

Anos 70 - Design de software e implicações no desenvolvimento

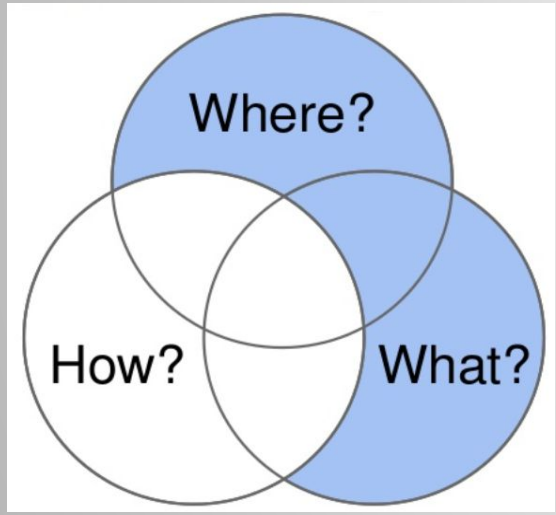
Anos 80 - Integração total do design no processo de desenvolvimento

- Referências à Arquitetura de Software começaram a surgir

1992 - Tópicos de Perry e Wolf

Ontem
Diferenças

Arquitetura de Software



Exemplos

Qual o tipo de armazenamento de dados?

Como os módulos se interagem entre si?

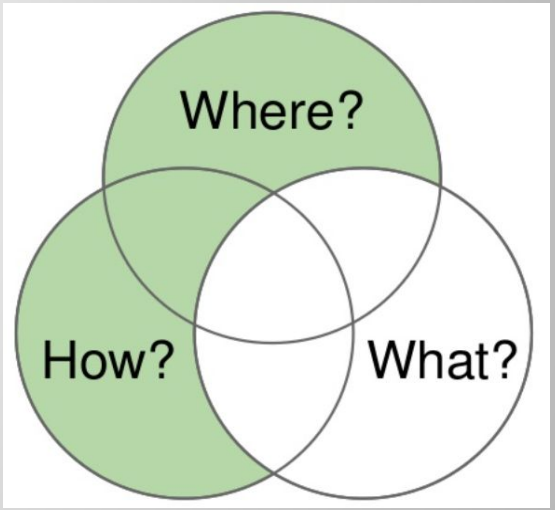
Qual sistema de recuperação está instalado?

Qual as responsabilidades do módulo X?

Quais as funções da classe Y?

O que uma classe pode ou não fazer?

Design de Software



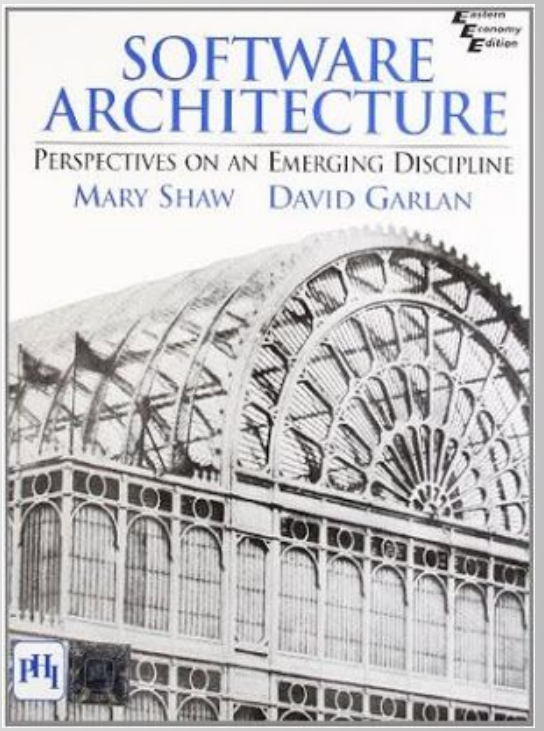
Ontem

Perspectives on an Emerging Discipline por Garlan and Shaw

Crescimento do número de padrões de arquiteturas de software (estilos)

Foi necessário uma classificação

Esse problema foi tratado no livro



Jan Bosch

Visão geral do estado da pesquisa atual na engenharia de software e na arquitetura de software

Ontem
Anos 80

Aparição da arquitetura de software nos anos 80

Disciplina madura com o uso de notações, ferramentas e várias técnicas

Pura e ocasional especulação, base da pesquisa acadêmica básica

Elemento essencial para a construção de software industrial

Ontem
Anos 90

Classic de Gamma

Padrões

Foi o primeiro a reunir e popularizar as ideias em grande escala.

Na era pré Gamma: Modelo MVC

Ontem
Computação Orientada a Serviço

Engenharia de software baseada em componentes (component-based software engineering (CBSE))

Melhor controle no design, implementação e na evolução de sistemas de software.

Transição voltada para o conceito do serviço e depois para a evolução natural para microsserviços.

Ontem
Computação Orientada a Serviço

Computação orientada a serviço (SOC) - Paradigma emergente para SDs e processamento de comércio eletrônico

Encontra sua origem na computação orientada a objetos e na computação de componentes.

Combate a complexidade de sistemas distribuídos e integra diferentes aplicações de software

Um programa que requisita um serviço fornece funcionalidades a outros componentes

Os serviços separam suas interfaces da própria implementação

Ontem
Computação Orientada a Serviço

Linguagens específicas são definidas para serem capazes de orquestrar as ações complexas dos serviços.

Utilizam alguns formalismos conhecidos da teoria concorrente, como o cálculo

Desenvolvimento de modelos formais

Melhor entendimento e verificação de interações de serviço

Ontem
Computação Orientada a Serviço

Benefícios:

- Dinamismo
- Modularização e reutilização
- Desenvolvimento distribuído
- Integração de sistemas heterogêneos e legado

Ontem

Segunda geração de serviços

Componentização na orientação de serviços (semelhante à POO)

Foco no encapsulamento

Cenário de memória compartilhada

Noção de capacidade de negócios

Foco na análise e no design para servir de base para que a arquitetura do sistema seja determinada.

Ontem

Segunda geração de serviços

A primeira “geração” de serviços orientados a arquitetura (SOA)

Assustadores contratos de serviço

Microsserviços são a evolução do conceito SOA e SOC

A ideia é retirar os níveis de complexidade desnecessários para focar na programação de serviços simples que implementam efetivamente uma única funcionalidade.

Ferramentas específicas para dar suporte aos programadores e naturalmente leva ao surgimento de padrões de designs específicos

Hoje

- Termo recente e utilizado para descrever um novo paradigma de programação

- Uma nova tendência em arquitetura de software

- Gerenciam a complexidade crescente decompondo grandes sistemas em um conjunto de serviços independentes.

Hoje

Diferenciais

- Tamanho
- Contexto Limitado
- Independência

Hoje

Características

- Flexibilidade
- Modularidade
- Evolução

Hoje

Equipes

- Organizar equipes multifuncionais em torno de serviços

- Abordagem “você constrói, você executa”

Hoje

Automação Total

- Entrega e atualização independentes

- Entrega e Integração Contínuas

Hoje

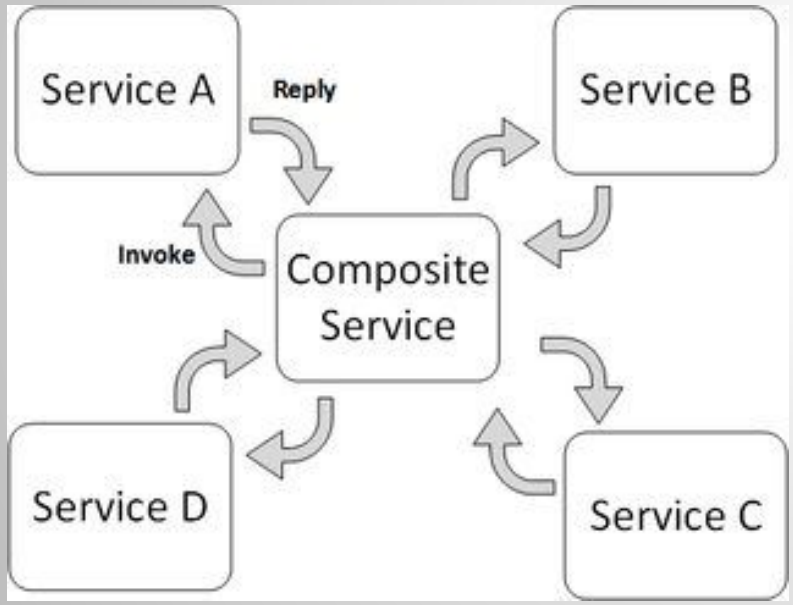
Orquestração x Coreografia

A orquestração conta com um serviço central (condutor) que gerencia as demandas e respostas dos serviços.

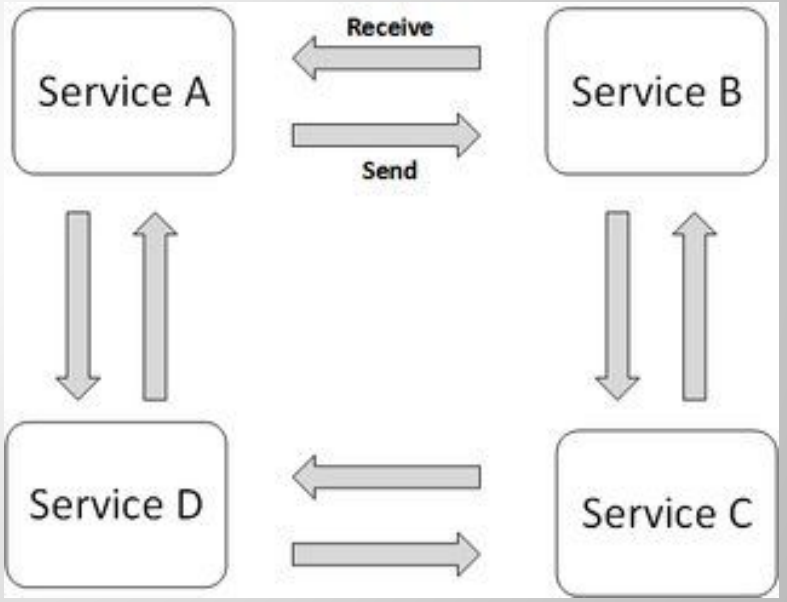
A coreografia utiliza eventos e mecanismos de publicação/inscrição para gerir a cooperação

Hoje

Orquestração



Coreografia



Hoje

Impacto na qualidade

Como os microsserviços impactam na qualidade do software?

Para isso vamos analisar os atributos:

- Disponibilidade (Availability)
- Confiabilidade (Reliability)
- Manutenibilidade (Maintainability)
- Desempenho (Performance)
- Segurança (Security)
- Testabilidade (Testability)

Hoje

Disponibilidade

- Avaliamos a disponibilidade de serviços independentes
  - Um único serviço indisponível pode gerar falha
  - Menor tamanho dos componentes = maior densidade de falhas
  - Maior tamanho dos componentes = maior tendência a falhas
  - Tamanho máximo dos microsserviços leva a tamanho ideal para reduzir falhas

- Criação de muitos microsserviços pode afetar a capacidade de disponibilizar todos simultaneamente e instantaneamente.

Hoje

Confiabilidade

- Confiabilidade é avaliada em serviços e mecanismos de troca de mensagens
- Construção em pequenos componentes

- O maior desafio é na integração
  - Troca de mensagens de forma direta

Hoje

Manutenibilidade

- Acoplamento fraco facilita manutenção e reduz os custos de:
  - Manutenção
  - Modificação de serviços
  - Correção de erros
  - Adição de novas funcionalidades

- Aspecto “Você constrói, você cuida” (“*you build it, you run it*”)

Hoje

Desempenho

- Fatores que afetam:
  - Comunicação
  - Restrições de tamanho
  - Qtd. de requisições na rede/memória
  - Grau de interconectividade
  - Acoplamento

Hoje

Segurança

Fatores que influenciam a segurança na arquitetura de microsserviços

- É necessário cuidado com a transferência de dados
- Criptografia
- Promovem reutilização de código
- Mecanismos de autenticação

Hoje

Testabilidade

A arquitetura de microsserviços permite:

- Teste em componentes isolados
- Ajustes no escopo dos testes
- Isolamento de mudanças e partes afetadas

A dificuldade está em testes de integração, principalmente em sistemas grandes

Amanhã

Pode-se tentar prever alguns possíveis problemas no futuro:

- Como gerenciar algumas mudanças em algum serviço que cause alguns efeitos negativos em outros serviços?
- Como evitar ataques que exploram falhas de rede?

Amanhã

Interface:

- Documentação informal;
- Tentando passar para uma documentação mais formal;
- Algumas tecnologias nem tem linguagens específicas ou checagem de compatibilidade com os microsserviços;
- Não existe nenhuma ferramenta suporte para checar se a interface do serviço implementado está correta.

Amanhã

Especificações comportamentais e coreografias:

- Mesmo definindo formalmente interfaces em forma de API não é o suficiente pra trazer compatibilidade, já que podem precisar realizar troca de mensagens em ordens específicas (deadlocks);
- Uma maneira de melhorar os microsserviços seria as execuções não-determinísticas.

Amanhã

Coreografia e Fundamentações Sólidas:

- Coreografias são descrições de alto nível de como as mensagens serão trocadas nos endpoints;
- Choreographic Programming;
- Todos esses pontos anteriormente mostram os problemas em especificação, verificação e sintetização em comportamentos de comunicações.

Amanhã

Segurança e Confiança

Ataques em Áreas extensas

- **Maior quantidade de Pontos de Acesso**
- **Heterogeneidade de Sistemas**
- **Exposição ao mundo externo**

Amanhã

Segurança e Confiança

Complexidade de Rede

- **Múltiplas comunicações**
- **Exposição à rede**
- **Complexidade de manutenção**

Amanhã

Segurança e Confiança

Confiança

- **Pressuposto de Comunicações íntegras**
- **Comprometimento de Serviços**
- **Operabilidade do Sistema**

Amanhã

Segurança e Confiança

Heterogeneidade

- Entidades Autônomas
- Vários Domínios
- Infraestrutura de Segurança
- Garantia de Aplicação de Regras

Conclusão