

ARQUITETURA DE SOFTWARE PDF

Neal Ford



Mais livros gratuitos no Bookey



Escanear para baixar

ARQUITETURA DE SOFTWARE

Dominando os Trade-offs das Decisões de
Arquitetura de Software Complexa

Escrito por Bookey

[Saiba mais sobre o resumo de ARQUITETURA DE
SOFTWARE](#)

[Ouvir ARQUITETURA DE SOFTWARE Audiobook](#)

Mais livros gratuitos no Bookey



Escanear para baixar

Sobre o livro

Navegar pelas complexidades da ARQUITETURA DE SOFTWARE não é uma tarefa fácil, pois os arquitetos frequentemente enfrentam decisões desafiadoras sem respostas claras. Neste livro perspicaz, os experientes especialistas Neal Ford, Mark Richards, Pramod Sadalage e Zhamak Dehghani compartilham seu vasto conhecimento sobre arquiteturas distribuídas, guiando os leitores através do intrincado panorama dos trade-offs arquiteturais. Por meio da narrativa envolvente de uma equipe fictícia — o Sysops Squad — este livro aprofunda-se em tópicos essenciais como granularidade de serviços, gestão de fluxos de trabalho, desacoplamento de contratos e transações distribuídas. Ao abordar dilemas arquiteturais comuns, ele equipa os leitores com técnicas práticas para avaliar e equilibrar os numerosos compromissos inerentes ao design de sistemas de software robustos, otimizando fatores críticos como escalabilidade, elasticidade e desempenho.

Mais livros gratuitos no Bookey



Escanear para baixar

Sobre o autor

Neal Ford é um destacado ARQUITETURA DE SOFTWARE e especialista em memes na ThoughtWorks, uma renomada consultoria de TI global especializada em desenvolvimento e entrega abrangente de software. Com uma vasta experiência em projetar e desenvolver aplicações, materiais didáticos e diversos meios de comunicação, ele é autor ou editor de cinco livros que cobrem uma ampla variedade de tecnologias. Neal é dedicado ao design e à construção de aplicações empresariais em larga escala e é reconhecido internacionalmente como palestrante, tendo realizado mais de 600 apresentações em mais de 100 conferências de desenvolvedores ao redor do mundo.

Mais livros gratuitos no Bookey



Escanear para baixar

Ad



Escanear para baixar



Experimente o aplicativo Bookey para ler mais de 1000 resumos dos melhores livros do mundo

Desbloqueie **1000+** títulos, **80+** tópicos

Novos títulos adicionados toda semana

Product & Brand

Liderança & Colaboração

Gerenciamento de Tempo

Relacionamento & Comunicação

Estratégia de Negócios

Criatividade

Memórias

Conheça a Si Mesmo

Psicologia

Empreendedorismo

História Mundial

Comunicação entre Pais e Filhos

Autocuidado

Mente

Visões dos melhores livros do mundo

Desenvolvimento

Os 7 Hábitos das Pessoas Altamente Eficazes



Mini Hábitos



Hábitos Atômicos



O Clube das 5 da Manhã



Como Fazer Amigos e Influenciar Pessoas



Como Não



Teste gratuito com Bookey



Lista de conteúdo do resumo

Capítulo 1 : 1. O Que Acontece Quando Não Há “Melhores Práticas”?

Capítulo 2 : I. Desmontando as Coisas

Capítulo 3 : 2. Discernindo o Acoplamento na

ARQUITETURA DE SOFTWARE

Capítulo 4 : 3. Modularidade Arquitetônica

Capítulo 5 : 4. Decomposição Arquitetônica

Capítulo 6 : 5. Padrões de Decomposição Baseados em Componentes

Capítulo 7 : 6. Desmembrando Dados Operacionais

Capítulo 8 : 7. Granularidade de Serviço

Capítulo 9 : II. Reunindo as peças

Capítulo 10 : 8. Padrões de Reuso

Capítulo 11 : 9. Propriedade de Dados e Transações Distribuídas

Mais livros gratuitos no Bookey



Escanear para baixar

Capítulo 12 : 10. Acesso a Dados Distribuídos

Capítulo 13 : 11. Gerenciando Fluxos de Trabalho

Distribuídos

Capítulo 14 : 12. Sagases Transacionais

Capítulo 15 : 13. Contratos

Capítulo 16 : 14. Gerenciando Dados Analíticos

Capítulo 17 : 15. Crie Sua Própria Análise de Compromissos

Capítulo 18 : A. Referências de Conceitos e Termos

Capítulo 19 : B. Referências de Registro de Decisões de
Arquitetura

Capítulo 20 : C. Referências de Trade-Off

Mais livros gratuitos no Bookey



Escanear para baixar

Capítulo 1 Resumo : 1. O Que Acontece Quando Não Há “Melhores Práticas”?



Seção	Resumo
Propósito dos Tecnólogos	Tecnólogos compartilham as "melhores práticas" da indústria, mas o termo é usado em demasia e surgem ceticismos, já que muitos desafios carecem de soluções claras, especialmente na ARQUITETURA DE SOFTWARE.
Desafios para Arquitetos	Arquitetos enfrentam desafios únicos ligados à sua organização, lidando frequentemente com problemas não documentados, tornando fútil a busca por soluções "milagrosas".
Tomada de Decisão na Arquitetura	Arquitetos devem focar nas compensações "menos ruins" ao invés das soluções "melhores", enfatizando a melhoria na tomada de decisões em situações novas.
Entendendo as "Partes Difíceis"	O título reflete tanto a dificuldade das decisões arquitetônicas quanto os elementos fundamentais da arquitetura que mudam com menos frequência do que os aspectos de design.
Conselhos Atemporais sobre ARQUITETURA DE SOFTWARE	O cenário de software em evolução e as tendências arquitetônicas anteriores destacam a necessidade de entender as restrições passadas que influenciaram as arquiteturas.
O Papel dos Dados na Arquitetura	Os dados devem ser projetados para durabilidade e seu valor mantido à medida que os sistemas evoluem. Sistemas fortemente acoplados e diferentes tipos de dados impactam o design arquitetônico.
Registros de Decisões Arquitetônicas (ADR)	ADRs documentam decisões arquitetônicas, fornecendo contexto e consequências, além de servir como uma ferramenta de comunicação no processo arquitetônico.
Funções de Aptidão Arquitetônica	Funções de aptidão automatizam a governança, garantindo a adesão aos princípios arquitetônicos e ilustrando a importância da validação contínua.
A Importância de Definições Claras	Definições claras de conceitos arquitetônicos-chave são cruciais para manter a clareza e evitar confusões nas discussões sobre arquitetura vs design.
Introduzindo a Saga da Equipe de Sysops	A saga acompanha a Penultimate Electronics enquanto navega desafios relacionados a suas aplicações monolíticas e requer uma evolução arquitetônica.
Conclusão	O capítulo molda a discussão sobre decisões arquitetônicas complicadas e implicações práticas através de narrativas do mundo real, estabelecendo uma base para futuras avaliações.



Capítulo 1: O Que Acontece Quando Não Há “Melhores Práticas”?

Propósito dos Tecnólogos

Os tecnólogos, como os arquitetos de software, apresentam-se em conferências ou escrevem livros para compartilhar o que são considerados "melhores práticas" na indústria. No entanto, o termo está se tornando excessivamente utilizado, levando ao ceticismo. Eles têm o objetivo de compartilhar soluções únicas para problemas comuns, mas muitos desafios não têm soluções claras, especialmente na ARQUITETURA DE SOFTWARE.

Desafios para Arquitetos

Diferente dos desenvolvedores, que muitas vezes conseguem encontrar soluções online para questões técnicas específicas, os arquitetos enfrentam desafios únicos que estão intimamente ligados ao contexto específico de sua organização. Eles lidam com problemas que podem não ter sido documentados ou abordados anteriormente, tornando a



busca por soluções "milagrosas" fútil—uma perspectiva ecoada por Fred Brooks.

Tomada de Decisão na Arquitetura

A essência do trabalho de um arquiteto reside em avaliar os trade-offs entre diversas decisões arquitetônicas. Os autores propõem uma abordagem humorística: ao invés de buscar a solução "melhor", os arquitetos devem buscar a combinação de trade-offs "menos ruim". O foco do livro está em melhorar os processos de tomada de decisão sob circunstâncias novas.

Compreendendo as “Partes Difíceis”

O título do livro, "ARQUITETURA DE SOFTWARE: As Partes Difíceis," reflete dois significados: a dificuldade inerente nas decisões arquitetônicas e o aspecto fundamental da arquitetura que muda menos frequentemente do que os elementos de design. A conversa sobre a definição da ARQUITETURA DE SOFTWARE muitas vezes resulta em debates improdutivos, porém é reconhecido que se trata de aspectos de um sistema que são difíceis de mudar.

Conselhos Atemporais sobre ARQUITETURA DE

Mais livros gratuitos no Bookey



Escanear para baixar

SOFTWARE

O cenário de software evolui continuamente, tornando difícil para os arquitetos manterem-se atualizados com estilos e práticas em mudança, como a transição de ARQUITETURAS orientadas a serviços para microserviços. Frequentemente, há a necessidade de entender as restrições que moldaram as tendências arquitetônicas anteriores, como a falta de soluções open source viáveis no passado.

O Papel dos Dados na Arquitetura

Os dados são cada vez mais críticos e devem ser projetados para durar mais do que os sistemas que servem. Os arquitetos devem garantir que os dados permaneçam valiosos à medida que os sistemas evoluem. Conflitos surgem quando sistemas fortemente acoplados são construídos, especialmente em ARQUITETURAS distribuídas. O livro aborda a importância dos diferentes tipos de dados, incluindo dados operacionais e analíticos, e suas implicações para o design.

Registros de Decisões Arquitetônicas (ADR)

Os ADRs servem como uma ferramenta para documentar



decisões arquitetônicas, permitindo uma reflexão e governança adicionais sobre essas decisões. Cada ADR documenta o contexto, a decisão e as consequências das escolhas arquitetônicas. Eles servem como uma ferramenta valiosa de comunicação ao longo do processo arquitetônico.

Funções de Aptidão Arquitetônica

As funções de aptidão ajudam a automatizar a governança na arquitetura, validando a conformidade com princípios arquitetônicos estabelecidos. Esse mecanismo permite que os arquitetos garantam que seus princípios sejam mantidos durante o processo de desenvolvimento sem supervisionar constantemente cada mudança. A importância da verificação contínua é reiterada com exemplos de ferramentas e práticas associadas.

A Importância de Definições Claras

Definir conceitos arquitetônicos-chave—como serviço, acoplamento e componente—é crucial para manter a clareza nas discussões sobre arquitetura versus design. Definições simples e claras ajudam a navegar pelas complexidades da arquitetura sem se perder em especificidades de



implementação.

Introduzindo a Saga da Sysops Squad

Para fundamentar as discussões em cenários do mundo real, o livro introduz a saga da Sysops Squad, que acompanha a Penultimate Electronics, uma empresa de eletrônicos que enfrenta desafios devido às suas aplicações monolíticas existentes e à necessidade de evolução arquitetônica.

Conclusão

O capítulo estabelece o cenário para entender decisões arquitetônicas complicadas, unindo teoria com implicações práticas através da narrativa da Sysops Squad e estabelecendo uma estrutura para avaliar e resolver desafios arquitetônicos no futuro.

Mais livros gratuitos no Bookey



Escanear para baixar

Exemplo

Ponto chave: Avaliação de Compromissos vs. Busca por Melhores Práticas

Exemplo: Como arquiteto, imagine que você foi encarregado de transformar uma aplicação crítica. Em vez de se deter em encontrar a única tecnologia ou abordagem 'melhor', concentre-se em entender os compromissos: Como cada escolha impacta a escalabilidade, manutenibilidade e as competências da equipe? Pense em escolher entre uma ARQUITETURA DE SOFTWARE de microservices e um design monolítico. Cada um tem suas vantagens e desvantagens, determinadas pelo contexto organizacional e pelas necessidades específicas, em vez de uma 'melhor prática' presumida pela indústria. Essa mentalidade promove uma tomada de decisão mais eficaz e realista, adaptada aos desafios reais.



Pensamento crítico

Ponto chave:Desafio em Encontrar 'Melhores Práticas'

Interpretação crítica:O ceticismo em relação às 'melhores práticas' na ARQUITETURA DE SOFTWARE destaca como os contextos únicos dos projetos muitas vezes tornam soluções generalizadas inaplicáveis. Enquanto Neal Ford enfatiza a tomada de decisões sob incerteza, isso leva os leitores a questionar se alguma prática pode ser rotulada como 'melhor' ou se esse termo representa apenas tendências passageiras da indústria. Apoiado por esse ceticismo, pesquisadores como Herbert Simon e seu conceito de 'racionalidade limitada' sugerem que as decisões são frequentemente tomadas com informações e contextos limitados, reforçando a necessidade de uma compreensão mais sutil das escolhas arquitetônicas.



Capítulo 2 Resumo : I. Desmontando as Coisas



Seção	Conteúdo
Parte I: Desmontando as Coisas	Discutindo a dissecação dos elementos da arquitetura para analisar as compensações.
Acoplamento Estático	Conexão de partes da arquitetura antes da execução, examinada em tempo de compilação, indica dependências.
Acoplamento Dinâmico	Como as partes da arquitetura interagem e se comunicam em tempo de execução.
Objetivo da Análise	Avaliar as compensações em arquiteturas distribuídas para uma melhor reconstrução do sistema.
Visão Geral dos Capítulos	<p>Capítulo 3: Modularidade e separação na arquitetura.</p> <p>Capítulo 4: Ferramentas para avaliar e desconstruir bases de código.</p> <p>Capítulo 5: Padrões para o design arquitetônico.</p> <p>Capítulo 6: Impacto de dados e transações na arquitetura.</p> <p>Capítulo 7: Acoplamento arquitetônico com preocupações de dados, focando nos limites de serviço.</p>
Definindo o Quantum da Arquitetura	Componentes independentes para implantação com alta coesão funcional, alto acoplamento estático e acoplamento dinâmico síncrono.
Características do Acoplamento Estático	Identifica dependências, auxilia na análise do impacto das mudanças nos componentes.
Acoplamento Quantum Dinâmico	<p>Comunicação: Síncrona vs. Assíncrona.</p> <p>Consistência: Transações atômicas estritas até a consistência eventual.</p> <p>Coordenação: Fluxos de trabalho orquestrados vs. coreografados.</p>



Seção	Conteúdo
Estrutura de Análise de Compensação	Identificar partes entrelaçadas, analisar acoplamento e avaliar os impactos de mudanças nos sistemas.
Conclusão	Fundamento para entender as complexidades em sistemas distribuídos e microserviços, focando no acoplamento e no quantum da arquitetura.

Parte I: Desmontando as Coisas

Entender uma arquitetura complexa envolve dissecar os elementos para analisar as compensações. Neal Ford discute dois tipos de acoplamento dentro da ARQUITETURA DE SOFTWARE: acoplamento estático e acoplamento dinâmico.

Acoplamento Estático vs. Acoplamento Dinâmico

-

Acoplamento Estático

: Refere-se a como as partes da arquitetura, como componentes e serviços, estão conectadas antes do tempo de execução. Pode ser examinado em tempo de compilação e indica as dependências dentro da arquitetura.

-

Acoplamento Dinâmico

: Descreve como essas partes interagem em tempo de



execução—como se comunicam e a natureza da troca de informações.

Objetivo da Análise

O foco principal é avaliar as compensações em arquiteturas distribuídas, permitindo uma compreensão detalhada antes de reconstruir todo o sistema.

Visão Geral dos Capítulos

-

Capítulo 3

: Discute modularidade e separação dentro das estruturas arquitetônicas.

-

Capítulo 4

: Introduz ferramentas para avaliar e desconstruir bases de código.

-

Capítulo 5

: Oferece padrões para auxiliar no design arquitetônico.

-

Capítulo 6

: Explora o impacto de dados e transações na arquitetura.

-

Capítulo 7

: Integra o acoplamento arquitetônico com preocupações de



dados, focando nos limites de serviço.

Definindo o Quantum da Arquitetura

Um quantum da arquitetura é um componente implantável de forma independente caracterizado por:

-

Alta Coesão Funcional

: Elementos da arquitetura funcionam juntos de forma eficaz, geralmente representando um domínio ou fluxo de trabalho.

-

Alto Acoplamento Estático

: Componentes estão intimamente conectados através de contratos e dependências.

-

Acoplamento Dinâmico Síncrono

: Representa a comunicação em tempo de execução.

Características do Acoplamento Estático

- Essencial para identificar dependências como frameworks, bibliotecas e outros requisitos operacionais.
- Ajuda na análise de como mudanças em uma parte afetam



outros componentes interconectados.

Acoplamento Quântico Dinâmico

O acoplamento dinâmico envolve vários aspectos da interação de serviços:

-

Comunicação

: Síncrona (aguardando uma resposta) vs. Assíncrona (continuando o trabalho sem esperar).

-

Consistência

: Varia de transações atômicas estritas a consistência eventual.

-

Coordenação

: Diferencia entre fluxos de trabalho orquestrados (controle central) e coreografados (interação descentralizada).

Estrutura de Análise de Compensações

Os arquitetos devem:

1. Identificar partes que estão entrelaçadas.
2. Analisar como estão acopladas.
3. Avaliar o impacto de mudanças em sistemas interdependentes.



Conclusão

Este capítulo estabelece as bases para entender as complexidades que os arquitetos enfrentam em sistemas distribuídos, especialmente em relação a microserviços. Ao esclarecer os conceitos de acoplamento estático e dinâmico e definir o quantum da arquitetura, os arquitetos podem navegar melhor pelas compensações e otimizar o design do sistema. Os capítulos futuros explorarão ainda mais o acoplamento dinâmico e seu impacto nos padrões de comunicação em microserviços.

Mais livros gratuitos no Bookey



Escanear para baixar

Exemplo

Ponto chave: Compreender as implicações do acoplamento estático e dinâmico é crucial para um design de sistema eficaz.

Exemplo: Imagine que você é um arquiteto de software encarregado de redesenhar uma grande plataforma de e-commerce. Ao começar a dissecar os componentes, você nota que o serviço de processamento de pagamentos está estaticamente acoplado à gestão de inventário, o que significa que mudanças em um podem impactar diretamente o outro. Essa conscientização permite que você decida estrategicamente se deve revisar o serviço de pagamento para reduzir as dependências ou implementar uma abordagem orientada a eventos para possibilitar a comunicação assíncrona. Ao dominar o acoplamento estático e dinâmico, você pode fazer escolhas arquitetônicas informadas que promovem resiliência e adaptabilidade em seu sistema.



Capítulo 3 Resumo : 2. Discernindo o Acoplamento na ARQUITETURA DE SOFTWARE

Seção	Resumo
Introdução	Discussões entre arquitetos revelam a pressão proveniente de um aplicativo de bilhetagem em falência e a necessidade de mudanças arquitetônicas para evitar interrupções nos negócios.
A Necessidade de Mudança	A liderança critica o gerenciamento do aplicativo de bilhetagem pela equipe de TI e enfatiza a urgência de resolver os problemas atuais e se adaptar às demandas futuras.
A Defesa da Modularidade	A ARQUITETURA DE SOFTWARE modular melhora a escalabilidade, a manutenibilidade e a tolerância a falhas ao permitir que as equipes se concentrem em componentes separados.
Fatores de Impulso para a Modularidade	Mudanças rápidas na concorrência e nas expectativas dos consumidores tornam necessárias mudanças nos sistemas de TI, especialmente durante fusões ou aquisições.
Benefícios da Modularidade	<p>Escalabilidade: Componentes modulares permitem crescimento sem necessidade de escalar todo o sistema.</p> <p>Agilidade: Adaptações rápidas às necessidades do negócio por meio de funcionalidades isoladas.</p> <p>Manutenibilidade: Mudanças ficam mais fáceis de implementar com menor necessidade de coordenação.</p> <p>Testabilidade: Testes independentes de módulos aumentam a precisão e a eficiência.</p> <p>Implantabilidade: Risco reduzido ao lançar atualizações leva a implantações mais frequentes e rápidas.</p>
Quadro Conceitual para a Mudança	Addison e Austen aproveitam a modularidade arquitetônica para apresentar um argumento forte aos patrocinadores do negócio, correlacionando problemas com benefícios modulares.
Preocupações com a Implementação	A transição para uma ARQUITETURA DE SOFTWARE distribuída pode incorrer em custos e desacelerar o desenvolvimento. Os riscos incluem problemas com o pipeline de implantação e a gestão de dependências de serviços.
Conclusão	A migração bem-sucedida para uma ARQUITETURA DE SOFTWARE modular depende de uma estratégia sólida, enfatizando a urgência de mudanças para resolver questões prementes e alinhar-se aos objetivos do negócio.

Capítulo 3: Modularidade Arquitetônica

Mais livros gratuitos no Bookey



Escanear para baixar

Introdução

A conversa entre os arquitetos na reunião da equipe Sysops exemplifica as tensões de uma aplicação de bilhetagem que não funciona. A liderança expressa frustração com problemas recorrentes, indicando uma necessidade urgente de mudança arquitetônica para evitar a interrupção de funções críticas do negócio. Addison e Austen, os arquitetos da aplicação, reconhecem a necessidade de quebrar a arquitetura monolítica para melhorar o desempenho, a manutenibilidade e a escalabilidade do sistema.

A Necessidade de Mudança

- Os líderes de negócios fazem uma crítica severa às dificuldades do departamento de TI com a aplicação de

**Instalar o aplicativo Bookey para desbloquear
texto completo e áudio**

Mais livros gratuitos no Bookey



Escanear para baixar



Escanear para baixar



Por que o Bookey é um aplicativo indispensável para amantes de livros



Conteúdo de 30min

Quanto mais profunda e clara for a interpretação que fornecemos, melhor será sua compreensão de cada título.



Clipes de Ideias de 3min

Impulsione seu progresso.



Questionário

Verifique se você dominou o que acabou de aprender.



E mais

Várias fontes, Caminhos em andamento, Coleções...

Teste gratuito com Bookey



Capítulo 4 Resumo : 3. Modularidade Arquitetônica

Resumo do Capítulo 4: Decomposição Arquitetônica

Contexto da Reunião

Addison e Austen têm a tarefa de migrar a aplicação monolítica do Sysops Squad para uma ARQUITETURA DE SOFTWARE distribuída devido a problemas de desempenho contínuos. Eles buscam determinar a melhor abordagem para a migração.

Abordagem para Decomposição

- Addison sugere começar pequeno, resolvendo problemas um passo de cada vez, lembrando o ditado "como você come um elefante?". Eles consideram iniciar pela função de relatórios, conhecida por causar travamentos, mas reconhecem a necessidade de abordar também os problemas de dados subjacentes.



- Eles consideram a base de conhecimento e as funcionalidades de pesquisa como possíveis pontos de partida, mas buscam uma abordagem mais metódica para decompor a aplicação.

Consulta com Logan

- Logan aconselha contra uma abordagem não estruturada, referindo-se a ela como "Antipadrão da Migração do Elefante", que frequentemente leva a um resultado caótico conhecido como "grande bola de lama distribuída."
- Em vez disso, ele apresenta dois métodos estruturados para decomposição arquitetônica:

decomposição baseada em componentes

e

forking tático

.

Decomposição Baseada em Componentes

- Este método foca em refinar e extrair componentes definidos gradualmente para estabelecer uma **ARQUITETURA DE SOFTWARE** distribuída.



- É mais eficaz quando os componentes existentes e os limites dos componentes são identificáveis dentro de uma base de código bem estruturada.

Forking Tático

- Esta abordagem pragmática envolve clonar o monólito existente e eliminar sistematicamente partes desnecessárias, permitindo que as equipes trabalhem de forma independente em serviços sem se preocupar com dependências.
- Mais adequado para bases de código mal estruturadas, que carecem de definições claras de componentes.

Avaliação da Estrutura do Código

- É importante que Addison e Austen avaliem sua base de código usando métricas como

abstratividade

e

instabilidade

de várias ARQUITETURAS DE SOFTWARE, determinando que é viável decompor sua aplicação.

- As descobertas revelam que, enquanto algum código alinha-se a estruturas ideais, outros existem em estados



caóticos, orientando-os a escolher o método de decomposição mais apropriado.

Tomada de Decisão

- Depois de pesar os prós e contras, Addison prefere a **decomposição baseada em componentes** devido aos limites de componentes estabelecidos e à redução das chances de código duplicado, o que complicaria a manutenção.
- Eles decidem contra o forking tático, preocupados com o potencial aumento de código duplicado enquanto mantêm uma estrutura interna deficiente, o que agravaria os problemas existentes.

Registro de Decisão de Arquitetura (ADR)

- Addison documenta a decisão de empregar a decomposição baseada em componentes, delineando os benefícios esperados, como aumento de confiabilidade, escalabilidade e agilidade na aplicação.
- Eles reconhecem que o esforço de migração pode levar mais tempo do que o forking tático, mas acreditam que essa abordagem oferece uma maneira mais controlada e



incremental de reescrever sua ARQUITETURA DE SOFTWARE.

Consequências da Migração

- Essa migração envolverá atrasos iniciais na liberação de novos recursos, aumento de custos e a potencial necessidade de reestruturar o banco de dados — tudo isso enquanto asseguram uma funcionalidade compartilhada coerente entre as fronteiras de serviço.

Em conclusão, o caminho metódico de Addison e Austen para decompor sua aplicação monolítica do Sysops Squad em uma ARQUITETURA DE SOFTWARE distribuída por meio da decomposição baseada em componentes destaca a importância de abordagens estruturadas na ARQUITETURA DE SOFTWARE.



Exemplo

Ponto chave: A importância de uma estratégia de decomposição estruturada.

Exemplo: Ao iniciar a migração de uma aplicação grande e complexa, imagine que você está navegando por um labirinto complicado, onde cada curva errada pode levar ao caos. Ao adotar um método estruturado, como a decomposição baseada em componentes, você garante que cada seção seja analisada e definida cuidadosamente, assim como mapear os caminhos do labirinto para encontrar a saída sem precisar voltar repetidamente. Essa abordagem gradual não apenas previne o risco de criar uma ‘bola de lama distribuída’ desorganizada, mas também permite que você identifique pontos problemáticos específicos e os trate de maneira estratégica, melhorando a escalabilidade e a manutenibilidade da sua aplicação a longo prazo.



Capítulo 5 Resumo : 4. Decomposição Arquitetônica

Seção	Descrição
Introdução	Apresenta a migração de Addison e Austen de uma arquitetura monolítica com orientação de Logan.
Visão Geral	Define uma sequência de padrões para dividir uma aplicação monolítica em componentes bem definidos.
Padrão de Identificação e Dimensionamento de Componentes	Catalogar e avaliar componentes para modularidade usando métricas como tamanho e ocupação de código.
Padrão de Coleta de Componentes de Domínio Comuns	Consolida a funcionalidade duplicada de domínio para reduzir sobreposições de serviços através da análise colaborativa.
Padrão de Desdobramento de Componentes	Elimina classes órfãs e refina a hierarquia de namespaces para clareza e integridade.
Padrão de Determinação de Dependências de Componentes	Avalia o acoplamento entre componentes para medir a complexidade da migração com ferramentas visuais.
Padrão de Criação de Domínios de Componentes	Agrupa componentes relacionados em domínios para estruturar a arquitetura de forma a facilitar a migração.
Padrão de Criação de Serviços de Domínio	Transfere domínios definidos em serviços separados, facilitando uma arquitetura baseada em serviços.
Histórias de Arquitetura	Registra e comunica as necessidades de refatoração que afetam a estrutura da aplicação com base nas necessidades do negócio.
Funções de Fitness para Governança	Funções automatizadas garantem conformidade com métricas de tamanho e estrutura durante toda a migração.
Conclusão	Uma abordagem sistemática usando padrões de decomposição minimiza riscos e prepara as equipes para futuras fases na decomposição de dados e ARQUITETURA DE SOFTWARE.

Capítulo 5: Padrões de Decomposição Baseados em Componentes

Mais livros gratuitos no Bookey



Escanear para baixar

Introdução

Addison e Austen adotaram a abordagem de decomposição baseada em componentes para sua migração de uma arquitetura monolítica. Eles buscaram orientação de Logan para entender os padrões de decomposição de forma eficaz.

Visão Geral dos Padrões de Decomposição Baseados em Componentes

Este capítulo descreve um conjunto de padrões e sua sequência para desmembrar uma aplicação monolítica, focando na criação de componentes bem definidos. Os principais padrões incluem:

1.

Padrão de Identificação e Dimensionamento de Componentes

: Identifica e dimensiona componentes para garantir uma arquitetura de aplicação modular.

2.

Padrão de Agrupamento de Componentes de Domínio Comum

: Consolida funcionalidades duplicadas de domínio para reduzir a sobreposição de serviços na arquitetura distribuída.



3.

Padrão de Achatar Componentes

: Garante que os componentes existam apenas como nós terminais em um namespace, eliminando hierarquias desnecessárias.

4.

Padrão de Determinação de Dependências de Componentes

: Analisa as dependências para avaliar o esforço necessário para a migração e identificar acoplamentos.

5.

Padrão de Criação de Domínios de Componentes

: Agrupa componentes em domínios lógicos para eventual separação em serviços.

6.

Padrão de Criação de Serviços de Domínio

: Move domínios de componentes bem definidos para serviços implantados separadamente.

Histórias de Arquitetura

Estas são usadas para registrar e comunicar as necessidades de refatoração que impactam a estrutura da aplicação com base nas necessidades de negócios.



Padrão de Identificação e Dimensionamento de Componentes

- Este padrão ajuda a catalogar e avaliar o tamanho dos componentes, visando aqueles que estão superdimensionados ou dimensionados de forma insuficiente.
- As métricas incluem nome do componente, namespace, percentual de código que ocupa, número de declarações e número de arquivos. O dimensionamento adequado é crucial para a modularidade.

Funções de Fitness para Governança

Funções automatizadas podem garantir conformidade contínua com as métricas de tamanho e estrutura dos componentes durante todo o processo de migração.

Padrão de Agrupamento de Componentes de Domínio Comum

- Este padrão ajuda a consolidar a lógica de negócios comum entre os componentes para eliminar redundâncias e reduzir a duplicação de serviços.



- Envolve a identificação de código compartilhado, que pode exigir análise colaborativa dos componentes existentes.

Padrão de Achatar Componentes

- O objetivo é eliminar classes órfãs que não pertencem a um componente bem definido, refinando a hierarquia do namespace para garantir clareza.
- Achatar adequadamente os namespaces ajuda a manter a integridade e a clareza dos componentes.

Padrão de Determinação de Dependências de Componentes

- Este processo analítico avalia o acoplamento entre componentes, informando a viabilidade e complexidade do projeto.
- A visualização das dependências ajuda a categorizar o esforço em segmentos gerenciáveis (dimensões de "bola de golfe", "bola de basquete" e "avião").

Padrão de Criação de Domínios de Componentes

- Uma vez que os componentes estão organizados, este



padrão ajuda a definir domínios para encapsular funcionalidades relacionadas e estruturar a arquitetura para facilitar a migração e a criação de serviços.

- Promove discussões colaborativas com as partes interessadas sobre o alinhamento de domínios e a refatoração necessária.

Padrão de Criação de Serviços de Domínio

- O passo final envolve a movimentação de grupos de domínios definidos para serviços separados, levando a uma arquitetura baseada em serviços.

- Esta movimentação é feita estrategicamente para evitar tumultos enquanto se garante a coesão dos serviços.

Conclusão

Uma abordagem sistemática usando esses padrões de decomposição minimiza os riscos na transição de arquiteturas monolíticas para distribuídas e orienta as equipes em direção a resultados bem-sucedidos. À medida que as equipes se envolvem nesse processo, também podem se preparar para abordar a decomposição de dados e a ARQUITETURA DE SOFTWARE em fases subsequentes.



Exemplo

Ponto chave: Identificar e dimensionar componentes é crucial para uma ARQUITETURA DE SOFTWARE modular.

Exemplo: Imagine que você está liderando um projeto de software onde diferentes equipes estão desenvolvendo funcionalidades de forma independente. Você percebe que alguns componentes são grandes demais, criando gargalos no desenvolvimento. Ao aplicar o 'Padrão de Identificação e Dimensionamento de Componentes', você analisa o tamanho de cada componente em termos de porcentagem de código e contagem de arquivos, identificando assim componentes excessivamente grandes que precisam ser divididos em menores e mais gerenciáveis. Isso permite que suas equipes trabalhem simultaneamente em diferentes aspectos da aplicação, melhorando a eficiência e promovendo uma ARQUITETURA DE SOFTWARE mais modular.



Pensamento crítico

Ponto chave: Análise Crítica dos Padrões de Decomposição Baseados em Componentes

Interpretação crítica: Embora o capítulo enfatize a abordagem estruturada para a transição de arquiteturas monolíticas para microserviços usando padrões de decomposição específicos, ele exige um exame crítico. A suposição de que um modelo único de decomposição de componentes leva a uma migração bem-sucedida ignora os desafios únicos apresentados por diferentes contextos organizacionais e sistemas legados. Nem todas as empresas poderão achar os padrões propostos adaptáveis sem complicações ou consequências indesejadas, como destacado por pesquisas de fontes como Bass et al. (2012), que enfatiza que os estilos arquitetônicos devem estar estreitamente alinhados tanto com as necessidades de negócios quanto com os ambientes técnicos. Os leitores devem abordar a perspectiva do autor com um olhar crítico, reconhecendo que desvios desses padrões podem ser necessários para atender a cenários específicos.



Capítulo 6 Resumo : 5. Padrões de Decomposição Baseados em Componentes

Capítulo 6: Padrões de Decomposição Baseados em Componentes

Introdução

Addison e Austen optaram por uma abordagem de decomposição baseada em componentes, mas não tinham clareza sobre seus padrões de decomposição, levando-os a consultar Logan em busca de orientação. Logan enfatizou a importância de executar cuidadosamente essa migração devido à sua visibilidade e relevância.

Visão Geral dos Padrões de Decomposição Baseados em Componentes

Este capítulo discute vários padrões-chave usados para refatorar aplicações monolíticas em componentes bem



definidos, que são pré-requisitos para a migração para arquiteturas distribuídas:

1.

Padrão de Identificação e Dimensionamento de Componentes

: Este padrão ajuda a identificar e gerenciar componentes, focando em seu tamanho para garantir que não sejam nem muito grandes nem muito pequenos.

- Componentes grandes muitas vezes levam a um alto acoplamento e dificultam a separação em serviços.
- Métricas como o total de declarações em um componente são introduzidas para o dimensionamento.

2.

Padrão de Reunião de Componentes de Domínio Comum

: Consolida a lógica de domínio comum para eliminar funcionalidades duplicadas, simplificando a criação de serviços.

**Instalar o aplicativo Bookey para desbloquear
texto completo e áudio**

Mais livros gratuitos no Bookey



Escanear para baixar

Ad



Escanear para baixar



App Store
Escolha dos Editores



22k avaliações de 5 estrelas

Feedback Positivo

Afonso Silva

...cada resumo de livro não só
..., mas também tornam o
...divertido e envolvente. O
...tizou a leitura para mim.

Fantástico!



Estou maravilhado com a variedade de livros e idiomas
que o Bookey suporta. Não é apenas um aplicativo, é
um portal para o conhecimento global. Além disso,
ganhar pontos para caridade é um grande bônus!

Brígida Santos

F



O
só
o
O

na Oliveira

...correr as
...ém me dá
...omprar a
...ar!

Adoro!



Usar o Bookey ajudou-me a cultivar um hábito de
leitura sem sobrecarregar minha agenda. O design do
aplicativo e suas funcionalidades são amigáveis,
tornando o crescimento intelectual acessível a todos.

Duarte Costa

Economiza tempo!



O Bookey é o meu apli
crescimento intelectual
perspicazes e lindame
um mundo de conheci

Aplicativo incrível!



Eu amo audiolivros, mas nem sempre tenho tempo para
ouvir o livro inteiro! O Bookey permite-me obter um resumo
dos destaques do livro que me interessa!!! Que ótimo
conceito!!! Altamente recomendado!

Estevão Pereira

Aplicativo lindo



Este aplicativo é um salva-vidas para
de livros com agendas lotadas. Os re
precisos, e os mapas mentais ajudar
o que aprendi. Altamente recomend

Teste gratuito com Bookey



Capítulo 7 Resumo : 6. Desmembrando Dados Operacionais

Resumo do Capítulo 7: Desmembrando Dados Operacionais

Introdução

Este capítulo discute os desafios e processos envolvidos na separação de uma base de dados monolítica em entidades menores e gerenciáveis, enfatizando a importância dos domínios de dados e a necessidade de diferentes tipos de bases de dados em certos cenários.

Compreendendo a Necessidade de Decomposição

- O aplicativo Sysops Squad passou por uma reformulação significativa, levando à percepção de que a base de dados monolítica precisava ser decomposta.
- Addison e Devon propuseram essa mudança, mas Dana, a arquiteta de dados, tinha reservas, citando a complexidade de



tal empreendimento.

Principais Fatores para a Decomposição de Dados

-

Controle de Mudanças

: Gerenciar alterações no esquema da base de dados é crucial, uma vez que modificações podem impactar múltiplos serviços que dependem da base de dados.

-

Gerenciamento de Conexões

: A capacidade das bases de dados de lidar com inúmeras conexões de múltiplos serviços é essencial para desempenho e escalabilidade.

-

Escalabilidade

: A capacidade da base de dados de escalar efetivamente com o aumento da demanda por serviços influencia diretamente o desempenho do sistema.

-

Tolerância a Falhas

: Uma base de dados compartilhada pode se tornar um ponto único de falha; portanto, dividi-la pode fortalecer a resiliência contra interrupções.



-

Quantum Arquitetural

: Manter alta coesão e baixo acoplamento em serviços em um nível arquitetural requer a separação das bases de dados.

-

Otimização do Tipo de Base de Dados

: Por meio do manuseio separado de vários tipos de dados, tipos de bases de dados mais adequados podem ser selecionados para necessidades específicas.

Técnicas para Decompor Dados

O capítulo introduz um processo de cinco etapas para desmembrar sistematicamente uma base de dados monolítica:

1.

Analisar a Base de Dados e Criar Domínios de Dados

: Identificar agrupamentos dentro da base de dados.

2.

Atribuir Tabelas a Domínios de Dados

: Categorizar tabelas pelo seu domínio e estabelecer esquemas.

3.

Separar Conexões de Base de Dados para os



Domínios de Dados

: Garantir que os serviços se conectem e acessem apenas seus esquemas designados.

4.

Mover Esquemas para Servidores de Base de Dados Separados

: Migrar domínios de dados para servidores individuais para otimizar as características operacionais.

5.

Migrar para Servidores de Base de Dados Independentes

: Finalizar transições redirecionando as conexões de serviços para os novos servidores.

Selecionando Tipos de Base de Dados Apropriados

A evolução das tecnologias de banco de dados trouxe vários tipos, como relacionais, NoSQL (documento, chave-valor, column-family e gráficos), NewSQL, e bases de dados nativas na nuvem. Cada tipo apresenta trocas únicas em curva de aprendizado, facilidade de modelagem de dados, escalabilidade, disponibilidade, consistência e suporte da comunidade.



Estudo de Caso: Saga Sysops Squad

A equipe justificou a necessidade de decomposição através de casos de uso específicos, como as altas demandas do segmento de relatórios afetando as funcionalidades de bilhetagem, o que levou a um argumento convincente para a segmentação da base de dados. No final, a decisão foi migrar certos domínios de dados (como pesquisas) para estruturas de base de dados mais adequadas (como bases de dados documentais) para melhorar a flexibilidade e a capacidade de resposta.

Conclusão

O capítulo destaca a complexidade de desmembrar uma base de dados monolítica, enfatizando uma abordagem sistemática para enfrentar esse desafio, considerando as várias trocas na **ARQUITETURA DE SOFTWARE**. Esta discussão detalhada serve como um guia para navegar pelos desafios da decomposição de dados.



Exemplo

Ponto chave: Importância da Decomposição de Bancos de Dados Monolíticos

Exemplo: Imagine que você está desenvolvendo uma aplicação complexa como parte de sua equipe. Você percebe como cada atualização desacelera drasticamente o sistema porque ele depende de um único banco de dados monolítico. Este gargalo restringe a inovação e a capacidade de resposta—cada vez que você deseja adicionar um recurso ou melhorar o desempenho, você enfrenta a intrincada teia de tabelas e serviços interconectados. Reconhecendo a necessidade de eficiência, sua equipe decide decompor o banco de dados em domínios menores e mais gerenciáveis. Ao criar esquemas direcionados e selecionar os tipos certos de bancos de dados para cada domínio de dados, você não apenas experimenta um aumento drástico na capacidade de resposta do sistema, mas também melhora sua capacidade de escalar eficientemente e aumentar a tolerância a falhas, garantindo que sua aplicação possa resistir melhor a falhas inesperadas. Esta mudança crucial transforma seu processo de desenvolvimento, permitindo uma iteração rápida e



crescimento.

Pensamento crítico

Ponto chave: A necessidade de decomposição de dados na ARQUITETURA DE SOFTWARE

Interpretação crítica: Enquanto o capítulo destaca as vantagens críticas de desmembrar um banco de dados monolítico, como escalabilidade e tolerância a falhas, é essencial reconhecer que essas soluções dependem do contexto. As necessidades específicas de um projeto podem nem sempre se alinhar com as sugestões do autor. Por exemplo, um estudo publicado no Journal of Systems and Software discute cenários em que manter um único banco de dados pode, na verdade, proporcionar um desempenho mais robusto devido à menor complexidade nas conexões e na gestão de transações (Smith et al., 2020). Portanto, os leitores devem avaliar criticamente as técnicas de decomposição sugeridas, levando em consideração os fatores únicos de seus ambientes operacionais específicos.



Capítulo 8 Resumo : 7. Granularidade de Serviço

Capítulo 8: Resumo da Granularidade de Serviço

Visão Geral da Granularidade de Serviço

A granularidade de serviço refere-se ao tamanho e escopo dos serviços individuais na ARQUITETURA DE SOFTWARE, um aspecto crucial ao fazer a transição para microserviços. Addison e Austen enfrentam dificuldades porque diversas opiniões sobre como decompor os serviços de domínio existentes levam à confusão.

Impacto das Decisões de Design

A equipe debate se deve consolidar os serviços em um único serviço ou dividi-los em menores. Os diálogos centrais se concentram em funcionalidades essenciais como bilhetagem, gerenciamento de clientes e separações de serviços. Por exemplo, Taylen defende serviços de responsabilidade única



e com granularidade fina, enquanto Addison e Austen são cautelosos quanto a dividir os serviços em demasia.

Desintegradores vs. Integradores de Granularidade

Entender quando desintegrar (quebrar serviços) ou integrar (combinar serviços) depende do reconhecimento de vários fatores:

-

Desintegradores:

-

Escopo e Função do Serviço:

Considere a coesão das funcionalidades. Se tarefas não relacionadas forem combinadas, pode ser necessário dividir.

-

Volatilidade de Código:

Mudanças frequentes em um aspecto de um serviço podem sugerir a sua separação para reduzir riscos de implantação.

-

Escalabilidade:

Quando diferentes partes precisam escalar independentemente, a separação pode melhorar o desempenho e os custos.



-

Tolerância a Falhas:

Serviços críticos para a operação devem ser separados para evitar que a falha de um componente afete todo o sistema.

-

Segurança:

Funcionalidades sensíveis podem precisar de mais cuidado do que outras e isso pode justificar um tratamento separado.

-

Extensibilidade:

Serviços que se espera que evoluam e se expandam podem se beneficiar de estarem segregados para permitir futuras mudanças sem grandes interrupções.

-

Integradores:

-

Transações de Banco de Dados:

Se transações em andamento requerem consistência entre bancos de dados, consolidar serviços pode ser necessário.

-

Fluxo de Trabalho e Coreografia:

Serviços que requerem comunicação podem complicar as dependências e os frameworks de desempenho.



-

Código Compartilhado:

Quando múltiplos serviços compartilham uma quantidade significativa de código, a sobrecarga aumenta; a consolidação pode simplificar a manutenção.

-

Relações de Dados:

Se as dependências de dados forem complexas, quebrar serviços pode complicar o acesso e exigir comunicação adicional entre os serviços.

Estudos de Caso em Granularidade de Serviço

1.

Sistema de Atribuição de Bilhetes

- Discussões sobre a atribuição e roteamento de bilhetes debatem se esses devem ser serviços separados ou um único serviço, com base no estreito acoplamento de suas funções. O resultado concluiu que um único serviço é preferível para desempenho e eficiência.

2.

Registro de Clientes



- A equipe considera ter um serviço consolidado para registro de clientes versus serviços separados por motivos de segurança. Após considerar a transacionalidade, optaram por um único serviço, garantindo segurança robusta por meio de bibliotecas especializadas.

Conclusão

Encontrar o equilíbrio certo na granularidade de serviço é um desafio contínuo. Os arquitetos devem analisar constantemente os trade-offs entre desintegração e integração, colaborando estreitamente com as partes interessadas nos negócios para definir estruturas de serviço ideais. Essa tomada de decisão cuidadosa depende da compreensão tanto dos desintegradores quanto dos integradores de granularidade para criar uma **ARQUITETURA DE SOFTWARE** eficaz de microserviços.



Pensamento crítico

Ponto chave: Equilíbrio da Granularidade do Serviço

Interpretação crítica: O capítulo destaca que a escolha entre a decomposição granular de serviços e a consolidação depende de vários fatores contextuais, como escalabilidade e segurança. No entanto, embora os argumentos do autor sejam convincentes, críticos podem afirmar que não existe uma solução única que sirva para todos na ARQUITETURA DE SOFTWARE, como evidenciado em obras como 'Microservices' de Martin Fowler e 'Building Microservices' de Sam Newman, que defendem uma compreensão mais sutil dos limites dos serviços.

Mais livros gratuitos no Bookey



Escanear para baixar

Capítulo 9 Resumo : II. Reunindo as peças

Parte II: Reunindo as peças

Esta seção do livro foca nos desafios de reconstruir arquiteturas distribuídas. Enfatiza que a divisão de módulos coesos pode levar a um aumento do acoplamento e a uma diminuição da legibilidade. Discute várias técnicas para gerenciar as complexidades em arquiteturas distribuídas, particularmente no que diz respeito à comunicação de serviços, contratos, fluxos de trabalho, transações e gerenciamento de dados.

Capítulo 8: Padrões de Reutilização

Neste capítulo, a narrativa gira em torno de uma equipe de desenvolvimento enfrentando conflitos sobre como lidar com código compartilhado enquanto realiza a transição para uma arquitetura distribuída. Dois membros da equipe, Taylen e Skyler, debatem sobre criar uma única DLL compartilhada versus bibliotecas compartilhadas separadas para sua



configuração distribuída. Um consenso é buscado ao examinar os prós e contras entre essas abordagens.

Reutilização de Código em Arquiteturas Distribuídas

Reutilizar código em sistemas distribuídos apresenta desafios únicos em comparação com arquiteturas monolíticas tradicionais. O capítulo destaca o ditado "reutilização é abuso", enfatizando uma abordagem cautelosa em relação ao compartilhamento de código em ambientes como microserviços.

Técnicas para Gerenciar a Reutilização de Código

1.

Replicação de Código

: Nesta abordagem, o código compartilhado é duplicado em

**Instalar o aplicativo Bookey para desbloquear
texto completo e áudio**

Mais livros gratuitos no Bookey



Escanear para baixar



Ler, Compartilhar, Empoderar

Conclua Seu Desafio de Leitura, Doe Livros para Crianças Africanas.

O Conceito



Esta atividade de doação de livros está sendo realizada em conjunto com a Books For Africa. Lançamos este projeto porque compartilhamos a mesma crença que a BFA: Para muitas crianças na África, o presente de livros é verdadeiramente um presente de esperança.

A Regra



Ganhe 100 pontos



Resgate um livro



Doe para a África

Seu aprendizado não traz apenas conhecimento, mas também permite que você ganhe pontos para causas beneficentes! Para cada 100 pontos ganhos, um livro será doado para a África.

Teste gratuito com Bookey



Capítulo 10 Resumo : 8. Padrões de Reuso

Resumo do Capítulo 10: Padrões de Reuso

Introdução

Na ARQUITETURA DE SOFTWARE distribuída, gerenciar funcionalidades compartilhadas torna-se complexo à medida que os desenvolvedores debatem os melhores métodos para reutilização de código e manejo de responsabilidades compartilhadas.

Personagens Principais no Debate

- Taylen defende múltiplas bibliotecas compartilhadas.
- Skyler propõe uma única DLL compartilhada para todos os serviços.
- Addison resolve conflitos por meio de discussões sobre compensações relacionadas à granularidade da biblioteca compartilhada e serviços compartilhados.



Técnicas de Reuso de Código

1.

Replicação de Código

- O código é copiado para o repositório de cada serviço para evitar compartilhamento.
- Comumente utilizado em microserviços iniciais como uma arquitetura "share nothing".
- Prós: Mantém o contexto delimitado, sem compartilhamento de código.
- Contras: Difícil de aplicar mudanças, sem controle de versão, levando à inconsistência.

2.

Biblioteca Compartilhada

- Uma biblioteca compartilhada é um artefato externo (por exemplo, JAR, DLL) utilizado por vários serviços.
- As compensações envolvem granularidade:
 -

Granularidade Grossa:

Simplifica o gerenciamento de dependências, mas complica o controle de mudanças.



-

Granularidade Fina:

Auxilia no controle de mudanças, mas torna o gerenciamento de dependências mais complexo.

-

Vantagens:

- O controle de versão melhora a agilidade para responder a mudanças.
- Baseado em compilação reduz erros em tempo de execução.

-

Desvantagens:

- Complexidade no gerenciamento de dependências, especialmente em grandes sistemas.
- A comunicação sobre mudanças de versão é frequentemente problemática.
- Dificuldades em estratégias de depreciação.

3.

Serviço Compartilhado

- Funcionalidade compartilhada é implementada como um serviço separado e implantado.



- Mudanças não exigem a reimplantação de serviços dependentes, mas aumentam os riscos em tempo de execução.

- Exige cuidadosa consideração do desempenho devido às chamadas de rede.

-

Vantagens:

- Reduz a duplicação de código entre vários serviços.

- Desacopla serviços, permitindo a implantação independente.

-

Desvantagens:

- Problemas de desempenho e tolerância a falhas devido a dependências de serviços externos.

- Aumento da complexidade no versionamento.

Sidecars e Service Mesh

- Propostos como uma solução para o acoplamento operacional em microserviços ao separar comportamentos operacionais da lógica de domínio.

-



Padrão Sidecar:

Um componente dedicado que lida com capacidades operacionais (por exemplo, registro, monitoramento) compartilhadas entre os serviços.

-

Service Mesh:

Uma camada de comunicação que habilita a comunicação entre serviços, integrando políticas para gerenciar microserviços.

Análise de Compensação

As principais compensações do padrão Sidecar incluem:

-

Vantagens:

Consistência nas operações, gerenciamento centralizado da infraestrutura.

-

Desvantagens:

Potencial complexidade na implementação e manutenção.

Conclusão

As estratégias de reuso de código devem ser avaliadas



através de compensações, enfocando tanto a taxa de mudança quanto a consistência operacional, em vez de simplesmente maximizar o reuso. O princípio orientador é garantir que o design suporte a agilidade do sistema, reduza o acoplamento e mantenha a integridade da ARQUITETURA DE SOFTWARE.

Mais livros gratuitos no Bookey



Escanear para baixar

Capítulo 11 Resumo : 9. Propriedade de Dados e Transações Distribuídas

Resumo do Capítulo 11: Propriedade de Dados e Transações Distribuídas

Introdução

Na ARQUITETURA DE SOFTWARE, definir a propriedade de dados e gerenciar transações distribuídas são desafios críticos, especialmente durante a transição de sistemas monolíticos para sistemas distribuídos. Este capítulo foca nesses conceitos, enfatizando a natureza colaborativa de determinar a propriedade dos dados entre os serviços.

Atribuição de Propriedade de Dados

A propriedade dos dados deve ser claramente atribuída aos serviços, o que não é simples devido a diferentes cenários de propriedade:

-



Propriedade Única

: Ocorre quando um serviço escreve exclusivamente em uma tabela. Este é o cenário mais simples, permitindo uma atribuição clara da propriedade.

-

Propriedade Comum

: Envolvem vários serviços escrevendo na mesma tabela, complicando as atribuições de propriedade.

-

Propriedade Conjunta

: Um cenário mais complexo onde apenas alguns serviços dentro do mesmo domínio escrevem na mesma tabela.

Cada cenário requer uma abordagem cuidadosa para atribuir a propriedade, minimizando problemas relacionados ao compartilhamento de dados e à consistência das transações.

Técnicas de Propriedade

1.

Propriedade Única

:

- Apenas um serviço escreve na tabela.
- Um modelo de propriedade claro, com o serviço de escrita possuindo os dados.



2.

Propriedade Comum

:

- Todos ou a maioria dos serviços requerem acesso de escrita a uma tabela.

- Um serviço dedicado é atribuído como o único proprietário, gerenciando operações de escrita, enquanto outros serviços enviam dados para este serviço.

3.

Técnicas de Propriedade Conjunta

:

-

Divisão de Tabela

: Dividindo uma tabela em várias tabelas para dar a cada serviço a propriedade sobre uma parte dos dados.

-

Domínio de Dados

: Criando um esquema de banco de dados compartilhado para vários serviços, complicando mudanças e exigindo coordenação entre eles.

-

Técnica de Delegação

: Atribuindo a propriedade a um serviço enquanto outros se comunicam para realizar operações de escrita através desse



serviço.

-

Consolidação de Serviços

: Unindo funcionalidades de múltiplos serviços em um único serviço para simplificar a propriedade. Isso pode aumentar os riscos de implantação e complicar a escalabilidade.

Transações Distribuídas

Transações são frequentemente pensadas em termos das propriedades ACID (Atomicidade, Consistência, Isolamento, Durabilidade), que garantem confiabilidade em contextos tradicionais. No entanto, quando as transações são distribuídas entre vários serviços, essas propriedades se tornam desafiadoras de serem aplicadas:

- O

Modelo BASE

é introduzido como uma alternativa representando Disponibilidade Básica, Estado Suave e Consistência Eventual. Diferente do ACID, a abordagem BASE permite que as transações mantenham métricas operacionais, mesmo que a consistência dos dados seja adiada.

Padrões de Consistência Eventual



Gerenciar transações distribuídas muitas vezes requer aceitar a consistência eventual. Os principais padrões discutidos incluem:

-

Sincronização em Segundo Plano

: Um processo para verificar e sincronizar dados periodicamente. É responsivo, mas pode acoplar fontes de dados e introduzir complexidade na lógica de negócios.

-

Padrão Baseado em Solicitações Orquestradas

: Um orquestrador gerencia solicitações, enfatizando a consistência dos dados sobre a responsividade, mas complicando o tratamento de erros.

-

Padrão Baseado em Eventos

: Usa mensagens assíncronas para que os serviços ajam em mudanças, promovendo desacoplamento, mas complicando o tratamento de erros, especialmente com falhas de mensagens.

Estudo de Caso: Processamento de Tickets

Discussão de um cenário prático onde equipes colaboraram para definir a propriedade dos dados para o processo de



conclusão de tickets e pesquisas, destacando os desafios da propriedade conjunta. Eles eventualmente concordaram com a técnica de delegação, com o Serviço de Pesquisa possuindo a tabela de Pesquisa e o Serviço de Conclusão de Tickets passando dados conforme necessário.

Conclusão

Este capítulo enfatizou as complexidades de atribuir a propriedade dos dados e gerenciar transações distribuídas dentro das arquiteturas modernas. Destacou a importância da colaboração e da escolha das técnicas de propriedade com base em casos de uso específicos para assegurar um design e operação de sistema eficazes.



Exemplo

Ponto chave: Adote técnicas colaborativas para a posse de dados em sistemas distribuídos.

Exemplo: Imagine que você está liderando uma equipe de desenvolvimento de software que está trabalhando em um novo sistema de gerenciamento de relacionamento com o cliente. Ao dividir a **ARQUITETURA DE SOFTWARE** em múltiplos serviços, você percebe que definir quem 'possui' os dados dos clientes não é simples. Em vez de atribuir a posse de maneira unilateral, você reúne sua equipe para discutir modelos potenciais de posse. Juntos, vocês exploram a noção de posse compartilhada, decidindo que o Serviço de Atendimento ao Cliente gerenciará exclusivamente os detalhes dos clientes, enquanto o Serviço de Pedidos terá acesso de leitura para processar os pedidos. Essa abordagem cooperativa não apenas esclarece as responsabilidades dos dados, mas também promove um senso de responsabilidade compartilhada e melhora a integridade geral do sistema.



Pensamento crítico

Ponto chave: A complexidade de atribuir propriedade de dados em sistemas distribuídos.

Interpretação crítica: Embora o autor enfatize a necessidade de definir uma clara propriedade de dados entre microserviços para garantir a confiabilidade do sistema, essa perspectiva pode ser excessivamente simplista. Na prática, a categorização rígida da propriedade de dados pode levar a gargalos e dificultar a agilidade, especialmente em ambientes que evoluem rapidamente. Críticos argumentam que modelos de propriedade excessivamente determinísticos podem sufocar a inovação e limitar a flexibilidade. Pesquisas sugerem que a adaptabilidade e o aprimoramento contínuo das práticas de propriedade, em vez da adesão estrita a categorias pré-definidas, podem resultar em melhores resultados em paisagens de software dinâmicas (veja fontes como os trabalhos de Martin Fowler sobre microserviços). Portanto, os leitores devem avaliar criticamente se tais estruturas determinísticas continuam relevantes ou aplicáveis em contextos variados.



Capítulo 12 Resumo : 10. Acesso a Dados Distribuídos

Padrão	Descrição	Vantagens	Desvantagens
Padrão de Comunicação Interserviços	Um serviço solicita dados de outro por meio de acesso remoto.	Simplicidade, sem problemas de volume de dados.	Latência de desempenho, desafios de escalabilidade, falta de tolerância a falhas.
Padrão de Replicação de Esquema de Coluna	Colunas da tabela de um serviço são duplicadas em outra.	Melhora no desempenho de leitura, latência reduzida.	Problemas de consistência de dados, possíveis conflitos de propriedade de dados, necessidade de sincronização.
Padrão de Cache Replicado	Caches em memória armazenam dados necessários por múltiplos serviços, sincronizados regularmente.	Maior capacidade de resposta, tolerância a falhas com dados prontamente disponíveis.	Dependência de inicialização no proprietário dos dados, potenciais desafios de sincronização, aumento nos requisitos de memória.
Padrão de Domínio de Dados	Um esquema de banco de dados compartilhado é utilizado por vários serviços.	Excelente desempenho, alta consistência de dados, integridade por meio de restrições de chave estrangeira.	Contexto de gerenciamento mais amplo, potenciais problemas de segurança com o acesso aos dados.

Capítulo 12: Acesso a Dados Distribuídos

Visão Geral dos Padrões de Acesso a Dados

Em uma arquitetura distribuída, acessar dados pertencentes a diferentes serviços pode ser desafiador. O capítulo discute quatro padrões principais para acesso a dados:

1. Padrão de Comunicação entre Serviços
2. Padrão de Replicação de Esquema de Coluna



3. Padrão de Cache Replicado

4. Padrão de Domínio de Dados

Cada padrão apresenta vantagens e desvantagens únicas, exigindo consideração cuidadosa das compensações.

Padrão de Comunicação entre Serviços

- Este é o padrão mais comum, onde um serviço solicita dados de outro por meio de acesso remoto.

-

Vantagens

: Simplicidade, ausência de problemas de volume de dados.

-

Desvantagens

: Latência de desempenho, desafios de escalabilidade e falta de tolerância a falhas.

Padrão de Replicação de Esquema de Coluna

**Instalar o aplicativo Bookey para desbloquear
texto completo e áudio**

Mais livros gratuitos no Bookey



Escanear para baixar



As melhores ideias do mundo desbloqueiam seu potencial

Essai gratuit avec Bookey



Escanear para baixar



Capítulo 13 Resumo : 11. Gerenciando Fluxos de Trabalho Distribuídos

Resumo do Capítulo 13: Gerenciando Fluxos de Trabalho Distribuídos

Introdução

Austen aborda Logan para oferecer ajuda em designs arquitetônicos, acreditando que a utilização da coreografia na nova arquitetura é necessária para manter os sistemas desacoplados. Logan o alerta contra absolutos no design arquitetônico e destaca a necessidade de análise de trade-offs em ARQUITETURA DE SOFTWARE distribuída moderna.

Forças de Acoplamento Chave

O capítulo identifica três forças principais que afetam os padrões de comunicação em ARQUITETURA DE SOFTWARE distribuída: comunicação, consistência e coordenação. Enfatiza a importância de coordenar serviços



para alcançar resultados desejados em fluxos de trabalho complexos.

Padrões de Coordenação

Dois padrões fundamentais de coordenação são discutidos: orquestração e coreografia.

Estilo de Comunicação de Orquestração

-

Definição

: Utiliza um orquestrador central para gerenciar estados de fluxo de trabalho, manter o tratamento de erros e coordenar atividades.

-

Vantagens

: O controle centralizado dos fluxos de trabalho simplifica lógicas complexas, auxilia no tratamento de erros e na gestão de estados, e permite a recuperabilidade.

-

Desvantagens

: Pode criar gargalos, pontos únicos de falha e desafios em escalabilidade.



Estilo de Comunicação de Coreografia

-

Definição

: Ao contrário da orquestração, a coreografia envolve controle descentralizado, onde os serviços se comunicam diretamente.

-

Vantagens

: Menos gargalos, maior capacidade de resposta e permite que os serviços escalam de forma independente.

-

Desvantagens

: Gerenciar erros e estados torna-se complexo sem um orquestrador central, aumentando a dificuldade dos fluxos de trabalho.

Análise de Trade-Offs

Os arquitetos devem ponderar os trade-offs entre orquestração e coreografia com base nos requisitos específicos do fluxo de trabalho:

- Fluxos de trabalho com alta complexidade podem se



beneficiar da orquestração para melhor tratamento de erros.

- Fluxos de trabalho mais simples podem ser melhor atendidos pela coreografia para melhorar o desempenho e a capacidade de resposta.

Gerenciamento de Estado de Fluxo de Trabalho

Gerenciar o estado de um fluxo de trabalho apresenta desafios, particularmente na coreografia devido à falta de um proprietário de estado central. Vários padrões para gerenciamento de estado são discutidos, incluindo:

1.

Padrão de Controlador Frontal

: Atribui o gerenciamento de estado ao primeiro serviço chamado.

2.

Coreografia Sem Estado

: Elimina o estado do fluxo de trabalho, mas aumenta a sobrecarga de comunicação.

3.

Acoplamento por Carimbo

: Armazena informações de estado dentro das mensagens, ajudando no rastreamento de fluxos de trabalho sem controle centralizado.



Sagas Transacionais

O capítulo define como as sagas transacionais podem ser implementadas usando vários padrões (por exemplo, Saga Épica, Saga de Contos de Fadas, etc.) e explora suas características, trade-offs e casos de uso apropriados. O conceito de transações compensatórias é crucial para restaurar os estados do sistema após falhas durante transações distribuídas.

Lições Chave

1.

Sem Soluções Absolutas

: Sempre avalie as necessidades específicas de um sistema em vez de se apegar estritamente a um padrão arquitetônico.

2.

O Gerenciamento de Erros é Crítico

: Tanto a orquestração quanto a coreografia têm implicações sobre como os erros são gerenciados, e é essencial levá-los em conta nas decisões de design.

3.

Consistência Eventual versus Atomicidade



: Compreenda os trade-offs entre manter forte consistência por meio de arquiteturas rigorosas versus usar consistência eventual para desempenho e escalabilidade.

Conclusão

A discussão incentiva engenheiros e arquitetos a reconhecer a complexidade e os trade-offs inerentes às ARQUITETURAS DE SOFTWARE distribuídas, particularmente no manuseio de fluxos de trabalho e gerenciamento de estado. Ao compreender e aplicar esses princípios, é possível projetar sistemas distribuídos robustos e responsivos.

Mais livros gratuitos no Bookey



Escanear para baixar

Pensamento crítico

Ponto chave:Compromissos nos Padrões Arquiteturais

Interpretação crítica:O capítulo enfatiza que os arquitetos devem analisar os compromissos entre orquestração e coreografia com base na complexidade específica do fluxo de trabalho. Embora a coreografia possa oferecer um desempenho melhor, ela carece de gerenciamento centralizado de erros em comparação com a orquestração, que simplifica a lógica, mas introduz gargalos. Essa visão sutil contrasta com a perspectiva do autor que favorece a coreografia para desacoplamento; é preciso considerar as complexidades do mundo real e o contexto em que essas metodologias são aplicadas. Vários estudos, como os encontrados em 'Designing Data-Intensive Applications' de Martin Kleppmann, ilustram que a aderência a estilos arquiteturais absolutos pode negligenciar requisitos situacionais.



Capítulo 14 Resumo : 12. Sagases Transacionais

Capítulo 14: Sagases Transacionais

Introdução

Austen visita Logan para discutir os feedbacks sobre o design do seu fluxo de trabalho de Bilhetagem, que Addison descreveu como uma ‘história de terror’. Logan esclarece que Austen se deparou com o infeliz design do padrão de saga História de Terror, um anti-padrão em ARQUITETURA DE SOFTWARE que lida com fluxos de trabalho transacionais.

Visão Geral dos Padrões de Saga Transacional

As sagas surgiram para limitar os escopos de bloqueio de banco de dados em arquiteturas distribuídas, evoluindo junto com microserviços. Elas representam sequências de transações locais onde eventos acionam atualizações posteriores. Em caso de falhas, atualizações compensatórias



revertem transações anteriores. Existem oito padrões diferentes de saga, cada um apresentando diferentes compensações.

Matiz de Padrões de Saga

Os oito padrões de saga podem ser entendidos através de uma matriz de três dimensões chave: comunicação (síncrona/assíncrona), consistência (atômica/eventual) e coordenação (orquestrada/coreografada).

1. Saga Épica (sao)
2. Saga de Telefone sem Fio (sac)
3. Saga de Conto de Fadas (seo)
4. Saga de Viagem no Tempo (sec)
5. Saga de Ficção Fantástica (aao)
6. História de Terror (aac)
7. Saga Paralela (aeo)
8. Saga Antológica (aec)

Saga Épica (sao)

- Características: Síncrona, Atômica, Orquestrada.
- Vantagens: Familiaridade para arquitetos monolíticos, imita modelos transacionais tradicionais.



- Desvantagens: Alto acoplamento, complexidade com transações compensatórias, gargalos que afetam o desempenho, problemas de escalabilidade.

Saga de Telefone sem Fio (sac)

- Características: Síncrona, Atômica, Coreografada.
- A coordenação é distribuída entre os serviços, evitando um único ponto de orquestração.
- Vantagens: Escalabilidade um pouco melhor devido à redução da orquestração, potencial para maior taxa de transferência.
- Desvantagens: Complexidade aumentada, mais tratamento de erros necessário nos serviços.

Saga de Conto de Fadas (seo)

- Características: Síncrona, Eventual, Orquestrada.
- Vantagens: Sem restrições de transação global, atualizações compensatórias mais fáceis.
- Desvantagens: Mantém alto acoplamento, o orquestrador ainda precisa lidar com fluxos de trabalho complexos.

Saga de Viagem no Tempo (sec)



- Características: Síncrona, Eventual, Coreografada.
- Oferece responsabilidades desacopladas entre serviços com possível fluxo de trabalho assíncrono.
- Melhor usada para fluxos de trabalho simples; a complexidade aumenta com a dificuldade do fluxo de trabalho.

Saga de Ficção Fantástica (aao)

- Características: Assíncrona, Atômica, Orquestrada.
- Aumenta a capacidade de resposta, mas apresenta complexidade significativa na coordenação e no tratamento de erros.

Saga História de Terror (aac)

- Características: Assíncrona, Atômica, Coreografada.
- A pior combinação; tenta manter a atomicidade, mas com acoplamento frouxo cria complexidade severa e riscos de erro.
- Não aconselhável devido à alta complexidade e baixa capacidade de resposta.



Saga Paralela (aeo)

- Características: Assíncrona, Eventual, Orquestrada.
- Melhoria em relação às sagas Épica e História de Terror.
- Coordenação simplificada com comunicação assíncrona melhora a escalabilidade e a capacidade de resposta.

Saga Antológica (aec)

- Características: Assíncrona, Eventual, Coreografada.
- Extremamente desacoplada, adequada para alta taxa de transferência e elasticidade.
- Alta complexidade devido à falta de orquestração a torna menos eficaz para fluxos de trabalho complexos.

Gerenciamento de Estado e Consistência Eventual

A consistência eventual e o gerenciamento de estado finito facilitam o tratamento de erros em sagas sem bloquear ações dos usuários. Esse método reduz o impacto nos usuários durante falhas e melhora a capacidade de resposta.

Conclusão



Os arquitetos devem entender as compensações associadas a cada padrão de saga. A escolha entre usar atualizações compensatórias versus gerenciamento de estado dependerá das necessidades específicas do sistema em relação à confiabilidade, desempenho e experiência do usuário. Implementar os padrões corretos pode afetar dramaticamente o sucesso das transações distribuídas dentro de uma ARQUITETURA DE SOFTWARE de microserviços.

Mais livros gratuitos no Bookey



Escanear para baixar

Capítulo 15 Resumo : 13. Contratos

Capítulo 13: Contratos

Introdução aos Contratos

- Addison e Sydney discutem métodos de comunicação para fluxos de trabalho de gestão de tickets, enfatizando a necessidade de acordos contratuais claros entre os serviços antes de escolhas de implementação como gRPC.

Tipos de Contratos

- Os contratos podem ser de duas formas:

contratos rígidos

, que exigem a adesão a detalhes específicos, e

contratos flexíveis

, que permitem mais liberdade.

- A escolha entre os tipos de contrato é influenciada por compensações na comunicação, consistência e coordenação.

- Os contratos podem incluir vários formatos, como SOAP, REST, gRPC, etc.



Contratos Rígidos vs. Flexíveis

-

Contratos Rígidos

: Exigem adesão exata a nomes, tipos e ordenação, levando a uma maior fragilidade, mas melhor documentação e fidelidade ao esquema.

- Prós: Fidelidade garantida ao contrato, mais fácil de verificar, melhor documentação.

- Contras: Emparelhamento rigoroso, gerenciamento de versões desafiador.

-

Contratos Flexíveis

: Oferecem flexibilidade, permitindo fácil evolução, mas trazem riscos de gestão como nomes mal escritos ou dados incompletos.

**Instalar o aplicativo Bookey para desbloquear
texto completo e áudio**

Mais livros gratuitos no Bookey



Escanear para baixar

Ad



Escanear para baixar




Experimente o aplicativo Bookey para ler mais de 1000 resumos dos melhores livros do mundo

Desbloqueie **1000+** títulos, **80+** tópicos

Novos títulos adicionados toda semana

Product & Brand

 Liderança & Colaboração

 Gerenciamento de Tempo

 Relacionamento & Comunicação

 Estratégia de Negócios

 Criatividade

 Memórias

 Conheça a Si Mesmo

 Psicologia

Empreendedorismo

 História Mundial

 Comunicação entre Pais e Filhos

 Autocuidado

 Mente

Visões dos melhores livros do mundo

amento
pos

Os 7 Hábitos das
Pessoas Altamente
Eficazes



Mini Hábitos



Hábitos Atômicos



O Clube das 5
da Manhã



Como Fazer Amigos
e Influenciar
Pessoas



Com
Não



Teste gratuito com Bookey



Capítulo 16 Resumo : 14. Gerenciando Dados Analíticos

Resumo do Capítulo 16: Gerenciando Dados Analíticos

Introdução

Logan e Dana discutem como lidar com dados analíticos em sua nova ARQUITETURA DE SOFTWARE, destacando a necessidade de um planejamento preditivo melhor utilizando ciência de dados extensa. Eles consideram implementar um data warehouse, mas descobrem suas limitações.

Abordagens Anteriores

1.

Data Warehouse

: ARQUITETURA DE SOFTWARE tradicional que separa dados operacionais e analíticos, mas apresenta vários desafios, como fragilidade, complexidade e problemas de



integração.

- As características incluem extração de dados de várias fontes, transformação em um único esquema e dados desnormalizados.

- As limitações observadas incluem a extrema partição do conhecimento de domínio e a complexidade envolvida na manutenção do warehouse.

2.

Data Lake

: Uma alternativa moderna que minimiza transformações iniciais, armazenando dados em seu formato bruto, apoiando assim ARQUITETURAS DE SOFTWARE mais dinâmicas e distribuídas.

- Embora menos estruturado que os warehouses, ainda enfrenta questões como descoberta de ativos, preocupações com privacidade e a não partição em domínios.

Data Mesh

O data mesh é apresentado como uma nova abordagem arquitetônica que alinha o gerenciamento de dados com os domínios de negócios para acesso e compartilhamento descentralizados.

-



Princípios do Data Mesh

:

-

Propriedade do Domínio sobre os Dados

: Os dados são gerenciados pelos domínios mais familiarizados, promovendo acesso peer-to-peer.

-

Dados como Produto

: Incentiva o compartilhamento e o design centrado no usuário de produtos de dados.

-

Plataforma de Dados Autônoma

: Oferece capacidades para desenvolvedores criando e encontrando produtos de dados facilmente.

-

Governança Computacional Federada

: Garante conformidade e governança através de um modelo federado envolvendo proprietários de produtos de dados de domínio.

Produto de Dados Quantum

- O elemento central no data mesh, alinhando análises com microsserviços. Cada domínio cria um

Mais livros gratuitos no Bookey



Escanear para baixar

Produto de Dados Quantum (PDQ)

que age independentemente enquanto permanece acoplado a sistemas operacionais.

- Os tipos de PDQs incluem PDQs alinhados à fonte, agregados e adequados para o propósito.

Compensações na Implementação do Data Mesh

- Compensações são estabelecidas entre dados operacionais e analíticos para avaliar as melhores práticas em ARQUITETURAS DE SOFTWARE distribuídas.
- Os desafios incluem a necessidade de uma governança contratual rigorosa, manutenção da consistência e garantia de comunicação eficiente entre os serviços.

Conclusão

Logan conclui que, embora a análise contínua de compensações seja crítica, não deve ser vista como um fardo, mas como uma oportunidade para desenvolver insights mais profundos sobre a ARQUITETURA DE SOFTWARE do sistema. Essa abordagem permite aprendizado e melhoria contínuos, garantindo que as equipes se concentrem em questões relevantes sem serem influenciadas por conselhos



genéricos.

Pensamentos Finais

A conversa ilustra a importância da colaboração, compreensão das compensações e a evolução contínua da **ARQUITETURA DE SOFTWARE** de dados para atender efetivamente às demandas analíticas modernas.

Mais livros gratuitos no Bookey



Escanear para baixar

Capítulo 17 Resumo : 15. Crie Sua Própria Análise de Compromissos

Resumo do Capítulo: Crie Sua Própria Análise de Compromissos

Introdução

O capítulo começa com uma reunião retrospectiva entre a Equipe de Operações de Sistemas, discutindo como o departamento de TI conseguiu reverter com sucesso uma linha de negócios falida. Os membros da equipe refletem sobre as lições valiosas aprendidas, especialmente a importância de entender os impulsionadores do negócio e a necessidade de colaboração entre as equipes de TI e de aplicações.

Principais Aprendizados da Reviravolta

1.

Compreensão dos Impulsionadores do Negócio

Mais livros gratuitos no Bookey



Escanear para baixar

: Priorizar os objetivos de negócio em vez de questões estritamente técnicas.

2.

Colaboração

: As equipes devem trabalhar juntas para abordar eficazmente os problemas nas migrações de aplicações e sistemas.

3.

Análise de Compromissos

: A análise adequada dos compromissos foi crucial para justificar decisões do ponto de vista empresarial.

Processo de Três Etapas para Análise de Compromissos

1.

Encontrando Dimensões Entrelaçadas

: Identificar aspectos da ARQUITETURA DE SOFTWARE que estão entrelaçados e se influenciam mutuamente.

2.

Analisando Acoplamento

: Compreender como os componentes dentro da ARQUITETURA DE SOFTWARE são interdependentes e como as mudanças afetam uns aos outros.

3.



Avaliando Compromissos

: Desenvolver uma estrutura robusta para entender o impacto das mudanças nos sistemas.

Encontrando Dimensões Entrelaçadas

- Um arquiteto deve descobrir os aspectos relacionados de sua ARQUITETURA DE SOFTWARE específica, que varia de um sistema para outro.

Análise de Acoplamento

- Defina acoplamento de forma simples: se uma alteração em um componente (X) exige uma mudança obrigatória em outro componente (Y).
- Crie diagramas de acoplamento estático para visualizar dependências e impactos operacionais.

Analisando Pontos de Acoplamento

- Um modelo leve de combinações permite que os arquitetos se concentrem nas forças significativas que precisam de atenção.
- As principais preocupações incluem acoplamento,



complexidade, capacidade de resposta/disponibilidade e escala/elasticidade.

Avaliando Compromissos

- Cenários contínuos de "e se" são essenciais para navegar pelas complexidades das decisões arquitetônicas.
- Os compromissos devem ser iterativos; uma vez que uma dimensão é decidida, outras são reavaliadas de acordo.

Técnicas para Análise de Compromissos

-

Qualitativo sobre Quantitativo

: Preferir avaliações qualitativas devido à natureza diversa das ARQUITETURAS DE SOFTWARE, incentivando uma análise qualitativa robusta para guiar decisões.

-

Listas MECE

: Utilizar listas mutuamente exclusivas e coletivamente exaustivas (MECE) para uma comparação eficaz de soluções alternativas.

Tomada de Decisão Contextual



- As decisões devem ser tomadas no contexto correto para evitar ser induzido em erro por soluções vantajosas que carecem de capacidades críticas. O contexto do mundo real deve orientar as **ARQUITETURAS DE SOFTWARE**.

Modelando Casos de Domínio Relevantes

- Os arquitetos devem derivar cenários particulares para ajudar a entender as implicações, focando em impulsionadores relevantes para filtrar opções de forma eficaz.

Comunicando Compromissos

- Ao discutir decisões arquitetônicas, reduzir a complexidade em pontos-chave ajuda a clarificar opções para partes interessadas não técnicas, enfatizando resultados em vez de detalhes complexos.

Evitando Evangelismo Tecnológico

- Os arquitetos devem evitar ficar excessivamente entusiasmados com tecnologias; análises profundas e



tomadas de decisão equilibradas são essenciais.

Conclusão

- Uma ARQUITETURA DE SOFTWARE bem-sucedida requer constante análise de compromissos adaptada a contextos únicos. Os arquitetos podem aproveitar testes iterativos para transitar de suposições qualitativas para insights concretos e quantitativos sobre sua ARQUITETURA DE SOFTWARE, melhorando a tomada de decisão e os resultados.

Mais livros gratuitos no Bookey



Escanear para baixar

Capítulo 18 Resumo : A. Referências de Conceitos e Termos

Apêndice A: Referências de Conceitos e Termos

Este apêndice fornece referências práticas a termos e conceitos fundamentais que são elaborados na obra anterior dos autores, "Fundamentos da ARQUITETURA DE SOFTWARE."

-

Complexidade ciclomática:

Capítulo 6, página 81

-

Acoplamento de componentes:

Capítulo 7, página 92

-

Cohesão de componentes:

Capítulo 7, página 93

-

Particionamento técnico versus de domínio:

Capítulo 8, página 103

-



Arquitetura em camadas:

Capítulo 10, página 135

-

Arquitetura baseada em serviços:

Capítulo 13, página 163

-

Arquitetura de micro serviços:

Capítulo 12, página 151

Apêndice B: Referências de Registro de Decisão de Arquitetura

Este apêndice consolida Registros de Decisão de Arquitetura (RDAs) que correspondem às decisões tomadas pela Equipe de Sysops ao longo do livro para fácil referência.

- RDA: Uma frase nominal curta contendo a decisão de arquitetura

- RDA: Migrar a Aplicação da Equipe de Sysops para uma

Instalar o aplicativo Bookey para desbloquear texto completo e áudio

Mais livros gratuitos no Bookey



Escanear para baixar



Escanear para baixar



Por que o Bookey é um aplicativo indispensável para amantes de livros



Conteúdo de 30min

Quanto mais profunda e clara for a interpretação que fornecemos, melhor será sua compreensão de cada título.



Clipes de Ideias de 3min

Impulsione seu progresso.



Questionário

Verifique se você dominou o que acabou de aprender.



E mais

Várias fontes, Caminhos em andamento, Coleções...

Teste gratuito com Bookey



Capítulo 19 Resumo : B. Referências de Registro de Decisões de Arquitetura

Referências de Registro de Decisões de Arquitetura

Esta seção consolida os Registros de Decisões de Arquitetura (ADR) correspondentes às decisões tomadas pela Equipe de Sysops, categorizadas por temas arquitetônicos específicos:

-

Formato Geral de Decisão

- “ADR: Uma frase nominal curta contendo a decisão arquitetônica”

-

ADRs Específicos

- Migrar a Aplicação da Equipe de Sysops para uma Arquitetura Distribuída

- Migração Usando a Abordagem de Decomposição Baseada em Componentes

- Uso de Banco de Dados Documental para Pesquisa de



Clientes

- Serviço Consolidado para Atribuição e Roteamento de Tickets
- Serviço Consolidado para Funcionalidades Relacionadas ao Cliente
- Uso de um Sidecar para Acoplamento Operacional
- Uso de uma Biblioteca Compartilhada para Lógica Comum de Banco de Dados de Tickets
- Propriedade de Tabela Única para Contextos Delimitados
- O Serviço de Pesquisa Possui a Tabela de Pesquisa
- Uso de Cache Replicado em Memória para Dados de Perfil de Especialista
- Uso de Orquestração para o Fluxo de Trabalho Primário de Tickets
- Contrato Flexível para o Aplicativo Móvel de Especialista da Equipe de Sysops

Referências de Compromissos

Este apêndice serve como um resumo para várias tabelas e figuras de compromissos que ilustram os trade-offs arquitetônicos discutidos no livro:

-

Tipos de Banco de Dados Avaliados

Mais livros gratuitos no Bookey



Escanear para baixar

- Bancos de dados relacionais
- Bancos de dados chave-valor
- Bancos de dados documentais
- Bancos de dados de famílias de colunas
- Bancos de dados de grafos
- Novos bancos de dados SQL
- Bancos de dados nativos da nuvem
- Bancos de dados de séries temporais

-

Compromissos de Técnica

- Técnica de replicação de código
- Técnica de biblioteca compartilhada
- Técnica de serviço compartilhado
- Técnica de sidecar/padrão de malha de serviços
- Técnicas de propriedade conjunta (divisão de tabela, domínio de dados, delegação, consolidação de serviços)
- Padrão de sincronização em segundo plano
- Padrão orquestrado baseado em requisições
- Padrão baseado em eventos
- Padrão de acesso a dados de comunicação entre serviços
- Padrão de acesso a dados de replicação de esquema de coluna



- Padrão de acesso a dados de cache replicado
- Padrão de acesso a dados de domínio de dados

-

Compromissos de Fluxo de Trabalho e Comunicação

- Padrões de orquestração e coreografia
- Tipos de contrato (estrito, flexível, orientado ao consumidor)
- Padrões de gerenciamento de dados (Data Warehouse, Data Lake, Data Mesh)
- Estilos de comunicação (síncrona vs. assíncrona, ponto a ponto vs. publicar e assinar)

Este resumo fornece acesso rápido às decisões arquitetônicas críticas e seus respectivos trade-offs discutidos no livro.

Mais livros gratuitos no Bookey



Escanear para baixar

Capítulo 20 Resumo : C. Referências de Trade-Off

Apêndice C: Referências de Trade-Off

O livro enfatiza a análise de trade-offs, acompanhada por várias tabelas e figuras na Parte II que resumem as preocupações arquitetônicas. Este apêndice fornece uma visão geral de todas as tabelas e figuras de trade-offs para referência rápida:

Figuras:

- Figura 6-25: Bancos de dados relacionais classificados segundo várias características de adoção
- Figura 6-26: Bancos de dados chave-valor classificados segundo várias características de adoção
- Figura 6-27: Bancos de dados de documentos classificados segundo várias características de adoção
- Figura 6-28: Bancos de dados de famílias de colunas classificados segundo várias características de adoção
- Figura 6-30: Bancos de dados gráficos classificados



segundo várias características de adoção

- Figura 6-31: Bancos de dados New SQL classificados

segundo várias características de adoção

- Figura 6-32: Bancos de dados nativos da nuvem classificados segundo várias características de adoção

- Figura 6-33: Bancos de dados de séries temporais classificados segundo várias características de adoção

Tabelas:

- Tabela 8-1: Trade-offs para a técnica de replicação de código

- Tabela 8-2: Trade-offs para a técnica de biblioteca compartilhada

- Tabela 8-3: Trade-offs para a técnica de serviço compartilhado

- Tabela 8-4: Trade-offs para a técnica de padrão Sidecar/rede de serviços

- Tabela 9-1: Trade-offs da técnica de divisão da tabela de propriedade conjunta

- Tabela 9-2: Trade-offs da técnica de domínio de dados de propriedade conjunta

- Tabela 9-3: Trade-offs da técnica de delegado de propriedade conjunta



- Tabela 9-4: Trade-offs da técnica de consolidação de serviço de propriedade conjunta
- Tabela 9-5: Trade-offs do padrão de sincronização em segundo plano
- Tabela 9-6: Trade-offs do padrão orquestrado baseado em solicitações
- Tabela 9-7: Trade-offs do padrão baseado em eventos
- Tabela 10-1: Trade-offs para o padrão de acesso a dados de Comunicação Interserviços
- Tabela 10-2: Trade-offs para o padrão de acesso a dados de Replicação de Esquema de Coluna
- Tabela 10-3: Trade-offs associados ao padrão de acesso a dados de cache replicado
- Tabela 10-4: Trade-offs associados ao padrão de acesso a dados de domínio de dados
- Tabela 11-1: Trade-offs para orquestração
- Tabela 11-2: Trade-offs para o padrão Front Controller
- Tabela 11-3: Trade-offs de coreografia sem estado
- Tabela 11-4: Trade-offs de acoplamento de carimbo
- Tabela 11-5: Trade-offs para o estilo de comunicação de coreografia
- Tabela 11-6: Trade-off entre orquestração e coreografia para fluxo de trabalho de ticket
- Tabela 11-7: Trade-offs atualizados entre orquestração e



coreografia para fluxo de trabalho de ticket

- Tabela 11-8: Trade-offs finais entre orquestração e coreografia para fluxo de trabalho de ticket
- Tabela 12-11: Trade-offs associados à gestão de estado versus transações distribuídas atômicas com atualizações compensatórias
- Tabela 12-12: Trade-offs associados a transações distribuídas atômicas e atualizações compensatórias
- Tabela 13-1: Trade-offs para contratos estritos
- Tabela 13-2: Trade-offs para contratos flexíveis
- Tabela 13-3: Trade-offs para contratos orientados ao consumidor
- Tabela 13-4: Trade-offs para acoplamento de carimbo
- Tabela 14-1: Trade-offs para o padrão de Data Warehouse
- Tabela 14-2: Trade-offs para o padrão de Data Lake
- Tabela 14-3: Trade-offs para o padrão de Data Mesh
- Tabela 15-2: Comparação consolidada de padrões de acoplamento dinâmico
- Tabela 15-3: Trade-offs entre comunicação síncrona e assíncrona para processamento de cartão de crédito
- Tabela 15-4: Trade-offs entre mensagens ponto a ponto versus publicar e assinar

Esta referência concisa tem como objetivo facilitar a compreensão dos vários trade-offs arquitetônicos explorados ao longo do livro.



Ad



Escanear para baixar



App Store
Escolha dos Editores



22k avaliações de 5 estrelas

Feedback Positivo

Afonso Silva

...cada resumo de livro não só
..., mas também tornam o
...divertido e envolvente. O
...tizou a leitura para mim.

Fantástico!



Estou maravilhado com a variedade de livros e idiomas
que o Bookey suporta. Não é apenas um aplicativo, é
um portal para o conhecimento global. Além disso,
ganhar pontos para caridade é um grande bônus!

Brígida Santos

FI



O
só
o
O

na Oliveira

...correr as
...ém me dá
...omprar a
...ar!

Adoro!



Usar o Bookey ajudou-me a cultivar um hábito de
leitura sem sobrecarregar minha agenda. O design do
aplicativo e suas funcionalidades são amigáveis,
tornando o crescimento intelectual acessível a todos.

Duarte Costa

Economiza tempo!



O Bookey é o meu apli
crescimento intelectual
perspicazes e lindame
um mundo de conheci

Aplicativo incrível!



Eu amo audiolivros, mas nem sempre tenho tempo para
ouvir o livro inteiro! O Bookey permite-me obter um resumo
dos destaques do livro que me interessa!!! Que ótimo
conceito!!! Altamente recomendado!

Estevão Pereira

Aplicativo lindo



Este aplicativo é um salva-vidas para
de livros com agendas lotadas. Os re
precisos, e os mapas mentais ajudar
o que aprendi. Altamente recomend

Teste gratuito com Bookey



Melhores frases do ARQUITETURA DE SOFTWARE por Neal Ford com números de página

Ver no site do Bookey e gerar imagens de citações bonitas

Capítulo 1 | Frases das páginas 155-597

1. Não tente encontrar o melhor design na ARQUITETURA DE SOFTWARE; em vez disso, esforce-se pela combinação menos ruim de compromissos.
2. Não há um único desenvolvimento, seja em tecnologia ou técnica de gestão, que por si só prometa mesmo uma ordem de magnitude [dez vezes] de melhoria em uma década em produtividade, confiabilidade e simplicidade.
3. Dados são uma coisa preciosa e durarão mais do que os próprios sistemas.
4. A segunda lei da ARQUITETURA DE SOFTWARE: o 'porquê' é mais importante do que o 'como'.
5. Funções de aptidão representam uma lista de verificação de princípios importantes definidos pelos arquitetos e

Mais livros gratuitos no Bookey



Escanear para baixar

executados como parte da construção para garantir que os desenvolvedores não os ignorem acidentalmente (ou propositalmente).

Capítulo 2 | Frases das páginas 598-661

1. Tudo é veneno, e nada está sem veneno; a dose sozinha faz com que uma coisa não seja um veneno." - Paracelsus
2. ARQUITETURA DE SOFTWARE é a coisa para a qual você não consegue encontrar respostas no Google.
3. Uma habilidade que os arquitetos modernos devem desenvolver é a capacidade de fazer análise de trade-offs.
4. Identifique quais partes estão entrelaçadas. Analise como estão acopladas umas às outras. Avalie os trade-offs determinando o impacto da mudança em sistemas interdependentes.

Capítulo 3 | Frases das páginas 662-968

1. Nenhuma dessas ferramentas ou abordagens é uma solução mágica. Arquiteturas distribuídas como microserviços são difíceis, especialmente se



os arquitetos não conseguem desvendar todas as forças em jogo.

2.Como muitas coisas, o acoplamento não é inerentemente ruim; os arquitetos só precisam saber como aplicá-lo de forma apropriada.

3.A ARQUITETURA DE SOFTWARE é a coisa que você não consegue encontrar no Google.

4.Para construir um bom caso de negócios para algo dessa magnitude, você precisa primeiro entender os benefícios da modularidade arquitetônica, associar esses benefícios às questões que você enfrenta com o sistema atual e, finalmente, analisar e documentar as compensações envolvidas na separação da aplicação.

5.As empresas devem ser ágeis para sobreviver no mercado volátil, rápido e em constante mudança de hoje, o que significa que as arquiteturas subjacentes também devem ser ágeis.

6.O esforço de migração incorrerá em custos adicionais (estimativas de custos a serem determinadas).



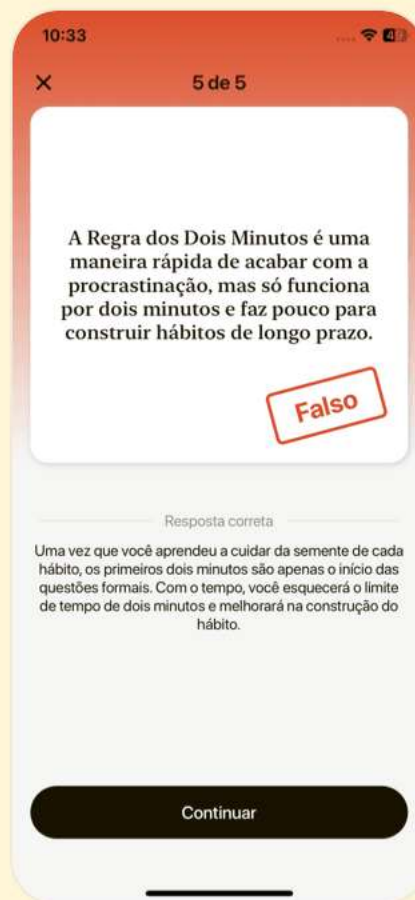
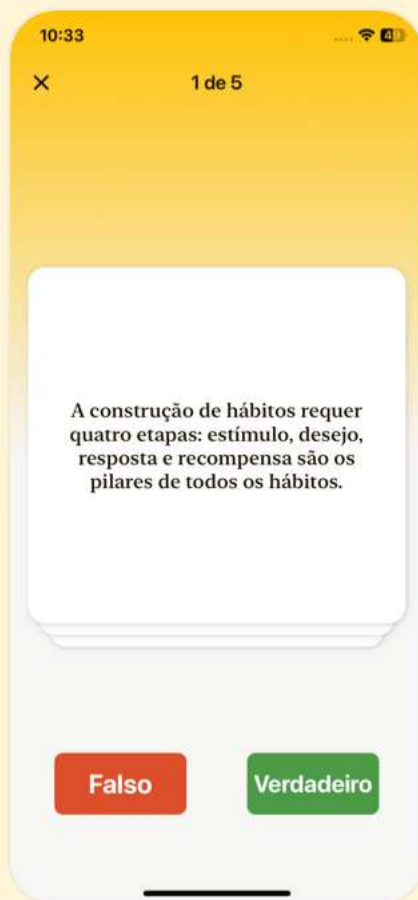


Baixe o app Bookey para desfrutar

Mais de 1000 resumos de livros com quizzes

Teste grátis disponível!

Escanear para baixar



Capítulo 4 | Frases das páginas 969-1245

1. Para construir um bom caso de negócios para algo dessa magnitude, você primeiro precisa entender os benefícios da modularidade arquitetônica, alinhar esses benefícios com os problemas que você está enfrentando com o sistema atual e, finalmente, analisar e documentar as compensações envolvidas em despedaçar a aplicação.
2. Se seus microserviços devem ser implantados como um conjunto completo em uma ordem específica, por favor, coloque-os de volta em um monólito e economize-se de algumas dores.
3. Enquanto a modularidade arquitetônica descreve o porquê de despedaçar uma aplicação monolítica, a decomposição arquitetônica descreve o como.
4. A abordagem que você está sugerindo é o que é conhecido como o Anti-Padrão da Migração do Elefante. Comer o elefante um pedaço de cada vez pode parecer uma boa estratégia no início, mas na maioria dos casos leva a uma



abordagem desestruturada que resulta em uma grande bola de lama distribuída.

5. Ao despedaçar aplicações monolíticas em ARQUITETURAS DE SOFTWARE distribuídas, construa serviços a partir de componentes, não de classes individuais.

Capítulo 5 | Frases das páginas 1246-1563

1. A aplicação é tão grande que eu nem sei por onde começar. É tão grande quanto um elefante!" exclamou Addison.
2. Como se come um elefante? Um pedaço de cada vez!
3. A abordagem que você está sugerindo é o que é conhecido como o Anti-Padrão da Migração do Elefante...leva a uma abordagem não estruturada que resulta em uma grande bola de lama distribuída, o que algumas pessoas também chamam de monólito distribuído.
4. Você precisa ter uma visão holística da aplicação e aplicar ou bifurcação tática ou decomposição baseada em componentes.



5. Essa abordagem reduz a chance de ter que manter código duplicado dentro de cada serviço.
6. O primeiro passo em qualquer migração monolítica é aplicar o padrão Identificar e Dimensionar Componentes.
7. O objetivo do padrão Determinar Dependências de Componentes é analisar as dependências de entrada e saída (acoplamento) entre componentes para determinar como ficaria o gráfico de dependência do serviço resultante após a separação da aplicação monolítica.
8. A nossa experiência mostra que esforços de migração 'na intuição' raramente produzem resultados positivos.
9. Ter um tamanho de componente relativamente consistente dentro de uma aplicação é importante.
10. É importante manter os componentes dentro de cada serviço de domínio alinhados com o domínio.

Capítulo 6 | Frases das páginas 1564-2804

1. Nós, bombeiros, precisamos nos unir. Eu já estive na sua posição antes, então sei como é voar às cegas nessas situações. Além disso, este é um



esforço de migração altamente visível, e é importante que vocês acertem da primeira vez.

Porque não haverá uma segunda chance.

2. Este capítulo apresenta um conjunto de padrões, conhecidos como padrões de decomposição baseados em componentes, que descrevem o refatoramento de código fonte monolítico para chegar a um conjunto de componentes bem definidos que podem eventualmente se tornar serviços.
3. É importante notar que este padrão diz respeito às dependências de componentes, e não às dependências de classes individuais dentro de um componente.
4. Identificar e entender o nível de acoplamento de componentes não apenas permite que o arquiteto determine a viabilidade do esforço de migração, mas também o que esperar em termos do nível geral de esforço.
5. Embora muitas ferramentas de análise estática de código possam mostrar o número de instruções dentro de um arquivo fonte, muitas delas não acumulam o total de



instruções por componente.

Mais livros gratuitos no Bookey



Escanear para baixar

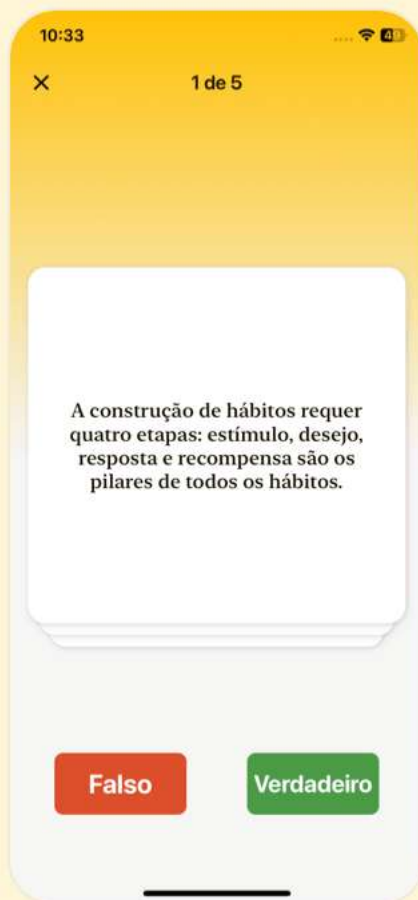


Baixe o app Bookey para desfrutar

Mais de 1000 resumos de livros com quizzes

Teste grátis disponível!

Escanear para baixar



Capítulo 7 | Frases das páginas 2805-4031

1. Dividir um banco de dados é difícil—muito mais difícil, na verdade, do que dividir a funcionalidade de uma aplicação.
2. Deadlocks e outros problemas transacionais se tornam mais difíceis de gerenciar quando os dados estão separados, já que múltiplos serviços podem competir para acessar os mesmos dados.
3. Uma abordagem eficaz é atribuir a cada serviço uma cota de conexão para governar a distribuição das conexões de banco de dados disponíveis entre os serviços.
4. Os arquitetos podem justificar um esforço de decomposição de dados ao entender e analisar desintegradores e integradores de dados.
5. Frequentemente, não todos os dados são tratados da mesma forma. Ao usar um banco de dados monolítico, todos os dados devem aderir a esse tipo de banco de dados, portanto, produzindo soluções potencialmente subótimas para certos tipos de dados.



- 6.O quântico arquitetônico ajuda a fornecer orientação em termos de quando dividir um banco de dados.
- 7.A soberania dos dados por serviço é o estado nirvana para uma ARQUITETURA DE SOFTWARE distribuída.
- 8.Dividir um banco de dados em contextos delimitados bem definidos ajuda significativamente a controlar as mudanças no banco de dados.
- 9.O perigo de serviços esquecidos é mitigado por uma análise de impacto diligente e um teste de regressão agressivo.
- 10.Convencem-me de que o banco de dados da equipe de Sysops realmente precisa ser separado. Forneça-me uma justificativa sólida.

Capítulo 8 | Frases das páginas 4032-4671

- 1.A chave para acertar na granularidade do serviço é remover opiniões e intuições, e usar desintegradores e integradores de granularidade para analisar objetivamente os trade-offs e formar justificativas sólidas sobre se deve ou não



desmembrar um serviço.

2. Nem toda parte de uma aplicação precisa ser microserviços.

Esse é um dos maiores erros do estilo de ARQUITETURA DE SOFTWARE de microserviços.

3. O segredo para acertar na granularidade é entender tanto os desintegradores de granularidade (quando desmembrar um serviço) quanto os integradores de granularidade (quando juntá-los de volta)...

4. Se esse for o caso, então como você determina quais serviços devem e não devem ser desmembrados?

5. Toda classe deve ter responsabilidade sobre uma única parte da funcionalidade desse programa, que deve encapsular. Todos os serviços desse módulo, classe ou função devem estar alinhados com essa responsabilidade.

6. O trade-off discutido em uma reunião com o proprietário do produto e o especialista em segurança é a transacionalidade versus segurança.

7. Precisamos desmembrar nosso serviço de pagamento para fornecer melhor extensibilidade para adicionar novos



métodos de pagamento...

Capítulo 9 | Frases das páginas 4672-4758

1. Tentar dividir um módulo coeso resultaria apenas em um aumento do acoplamento e na diminuição da legibilidade.
2. Uma vez que um sistema é fragmentado, os arquitetos frequentemente acham necessário reconstitui-lo para que funcione como uma unidade coesa.
3. A reutilização de código é uma parte normal do desenvolvimento de software.
4. Embora os desenvolvedores devam tentar limitar a quantidade de reutilização de código em ARQUITETURAS DISTRIBUÍDAS, é, no entanto, um fato da vida no desenvolvimento de software e deve ser abordado, particularmente em ARQUITETURAS DISTRIBUÍDAS.
5. A escolha da granularidade da biblioteca compartilhada pode não importar muito com apenas alguns serviços, mas à medida que o número de serviços aumenta, também



aumentam os problemas associados ao controle de mudanças e à gestão de dependências.

Mais livros gratuitos no Bookey



Escanear para baixar

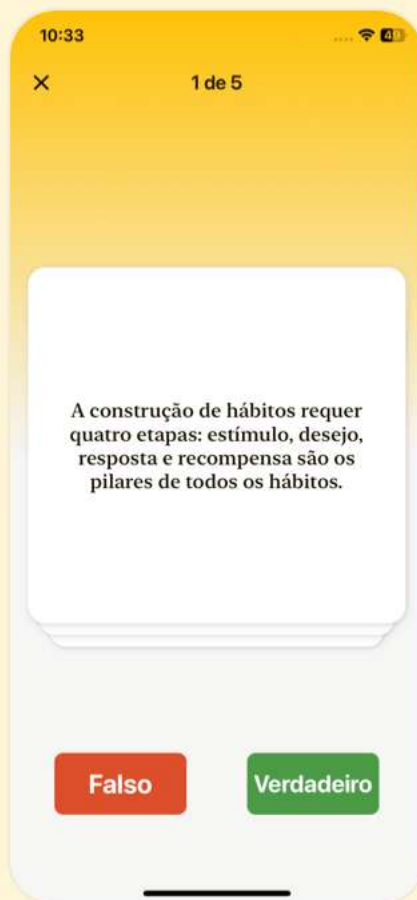


Baixe o app Bookey para desfrutar

Mais de 1000 resumos de livros com quizzes

Teste grátis disponível!

Escanear para baixar



Capítulo 10 | Frases das páginas 4759-5493

1. A reutilização é derivada da abstração, mas operacionalizada por uma taxa lenta de mudança.
2. A reutilização de código é uma parte normal do desenvolvimento de software... em arquiteturas distribuídas, as coisas se tornam um pouco mais complicadas.
3. A questão é que, com a técnica de serviço compartilhado, mudanças em um serviço compartilhado são geralmente de natureza em tempo de execução e, portanto, carregam muito mais risco do que com bibliotecas compartilhadas.
4. Versionar suas bibliotecas compartilhadas fornece não apenas compatibilidade retroativa, mas também um alto nível de agilidade— a capacidade de responder rapidamente a mudanças.
- 5....se uma equipe altera o código compartilhado, todas as equipes devem coordenar-se com essa mudança.

Capítulo 11 | Frases das páginas 5494-6275

1. Após conversar com Dana e aprender sobre a



propriedade dos dados e a gestão de transações distribuídas...

- 2.O serviço que realiza operações de escrita na tabela de dados é o proprietário dessa tabela...
- 3.Quando apenas um serviço escreve em uma tabela, essa tabela será atribuída à propriedade desse serviço.
- 4.Infelizmente, atribuir a propriedade dos dados a um serviço não é tão simples quanto parece...
- 5.A regra geral para atribuir a propriedade da tabela afirma que os serviços que realizam operações de escrita em uma tabela são os proprietários dessa tabela.
- 6.Embora o compartilhamento de dados seja geralmente desencorajado em arquiteturas distribuídas, isso resolve alguns problemas de desempenho, disponibilidade e consistência de dados...
- 7.A abordagem de delegação discutida na seção anterior destaca o problema principal associado à propriedade conjunta—dependência de serviço.
- 8.Adicionar um serviço de orquestração dedicado não apenas



acrescenta saltos adicionais na rede...

- 9.O padrão baseado em eventos utiliza mensagens assíncronas de publicação e assinatura ou fluxos de eventos para alcançar consistência eventual.
- 10.Isso exigiria que o orquestrador tivesse todas as informações necessárias para reinserir o cliente...

Capítulo 12 | Frases das páginas 6276-6490

- 1.Sempre é um termo complicado em ARQUITETURA DE SOFTWARE. Eu tive um mentor que tinha uma perspectiva memorável sobre isso... nunca diga nunca. Não consigo pensar em muitas decisões na arquitetura onde sempre ou nunca se aplica.
- 2.Um dos principais objetivos do estilo de ARQUITETURA DE MICROSERVIÇOS é o desacoplamento, e usar um componente global como um ESB cria um ponto de acoplamento indesejável.
- 3.Os benefícios da gestão de estado descentralizada na coreografia são reatividade, escalabilidade e redução do



acoplamento de serviços...

4. Se o arquiteto organizou sua arquitetura da mesma forma que os domínios, a implementação do fluxo de trabalho deve ter uma complexidade semelhante.

5. A primeira lei da ARQUITETURA DE SOFTWARE afirma... tudo na ARQUITETURA DE SOFTWARE é um trade-off...

Mais livros gratuitos no Bookey



Escanear para baixar

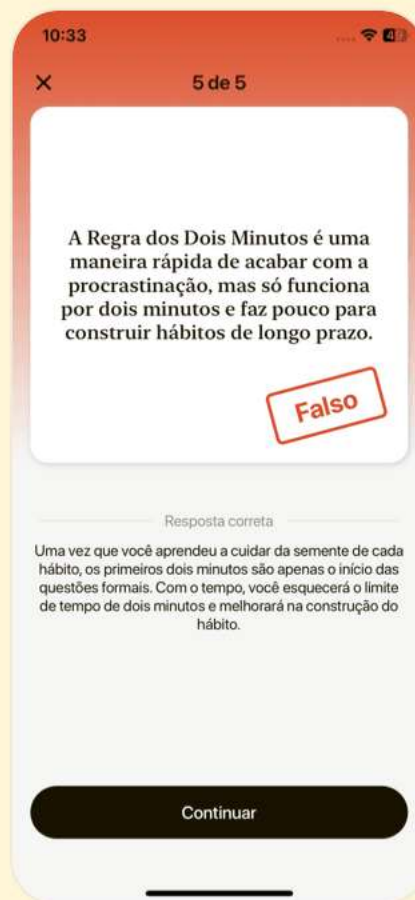
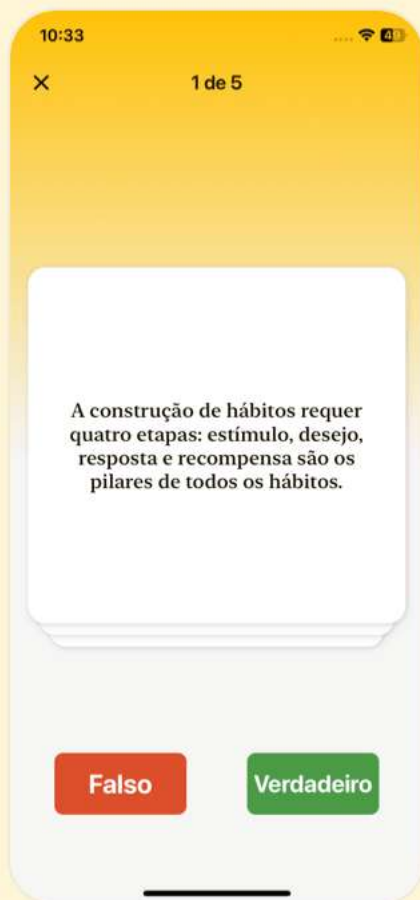


Baixe o app Bookey para desfrutar

Mais de 1000 resumos de livros com quizzes

Teste grátis disponível!

Escanear para baixar



Capítulo 13 | Frases das páginas 6491-6781

1. Sempre é um termo complicado em **ARQUITETURA DE SOFTWARE**. Eu tive um mentor que tinha uma perspectiva memorável sobre isso, e que sempre dizia: Nunca use absolutos ao falar sobre arquitetura, exceto quando falar sobre absolutos.
2. Um arquiteto nunca pode reduzir o acoplamento semântico por meio da implementação, mas pode torná-lo pior.
3. Se o mundo consistisse apenas em caminhos felizes, a **ARQUITETURA DE SOFTWARE** seria fácil.
4. A clara vantagem do Epic Saga(sao) é a coordenação transacional que imita sistemas monolíticos, juntamente com o claro proprietário do fluxo de trabalho representado por meio de um orquestrador.
5. Muito do mundo real não é transacional, como observado no famoso ensaio de Gregor Hohpe, 'Starbucks Não Usa Compromisso de Duas Fases.'
6. Adicionar assincronidade a fluxos de trabalho orquestrados



adiciona um estado transacional assíncrono à equação, removendo suposições seriais sobre a ordenação e adicionando as possibilidades de deadlocks, condições de corrida, e uma série de outros desafios de sistemas paralelos.

7. Este é o problema com transações distribuídas atômicas—o usuário final está desnecessariamente acoplado semanticamente ao processo de negócios.
8. As questões em torno de atualizações compensatórias são complexas e podem levar a cenários onde ações realizadas por outros serviços utilizando dados da atualização anterior podem já ter ocorrido e podem não ser reversíveis.

Capítulo 14 | Frases das páginas 6782-7530

1. Muitos arquitetos iniciantes ou ingênuos acreditam que, porque um padrão existe para um problema, ele representa uma solução limpa. No entanto, o padrão é apenas um reconhecimento de uma commonalidade, não de uma solucionabilidade.



- 2.A comunicação assíncrona melhora o desempenho. No entanto, como arquitetos, não podemos considerá-la isoladamente quando está entrelaçada com outras dimensões da arquitetura, como consistência e coordenação.
- 3.A coordenação de transações é uma das partes mais difíceis da arquitetura, e quanto mais amplo for o escopo, pior se torna.
- 4.Nós chamamos isso de efeito colateral dentro de arquiteturas distribuídas. Ao reverter a transação no Serviço de Ingressos, ações realizadas por outros serviços utilizando dados da atualização anterior podem já ter ocorrido e podem não ser reversíveis.
- 5.Se tivermos problemas com atomicidade, podemos investigar esses padrões como alternativas.

Capítulo 15 | Frases das páginas 7531-7771

- 1.Bem, isso é uma implementação, não uma arquitetura”, disse Addison. “Precisamos decidir quais tipos de contratos queremos antes de



escolher como implementá-los.

2. Manter os contratos em um nível de ‘necessidade de saber’ cria um equilíbrio entre acoplamento semântico e informação necessária, sem criar fragilidade desnecessária na ARQUITETURA DE SOFTWARE de integração.
3. Contratos orientados ao consumidor permitem que o provedor e os consumidores se mantenham sincronizados por meio de governança arquitetônica automatizada.
4. Um padrão antitípico comum que alguns arquitetos caem na armadilha é assumir que a Wishlist pode eventualmente precisar de todas as outras partes, então os arquitetos as incluem no contrato desde o início.
5. Usar contratos flexíveis permite sistemas extremamente desacoplados, frequentemente um dos objetivos em ARQUITETURAS DE SOFTWARE, como microserviços.



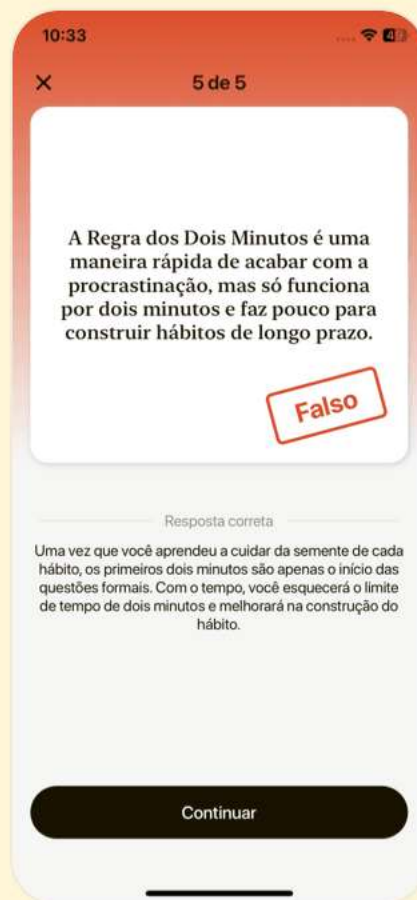
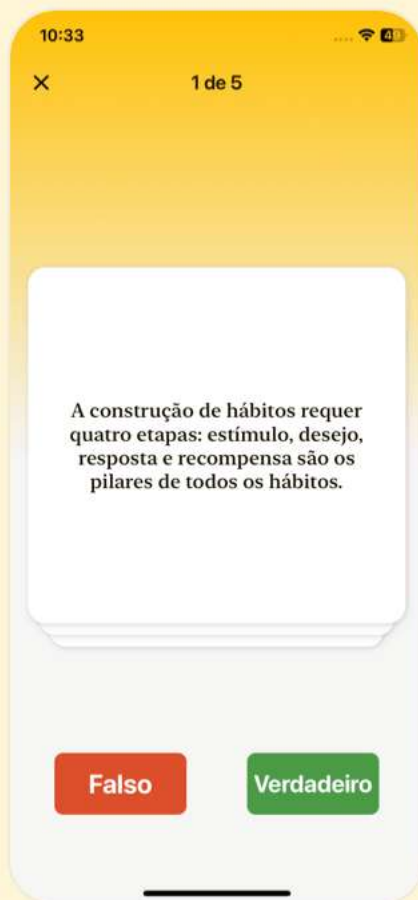


Baixe o app Bookey para desfrutar

Mais de 1000 resumos de livros com quizzes

Teste grátis disponível!

Escanear para baixar



Capítulo 16 | Frases das páginas 7772-8114

1. A suposição básica era que os dados operacionais eram armazenados em bancos de dados relacionais acessíveis diretamente pela rede.
2. A maquinaria do data warehouse normalmente apresentava armazenamento e computação massivamente capazes, aliviando os pesados requisitos para seu próprio ecossistema.
3. Os dados são possuídos e compartilhados pelos domínios que estão mais intimamente familiarizados com os dados: os domínios que ou originam os dados, ou são os consumidores de primeira classe dos dados.
4. O data mesh introduz o conceito de dados servidos como um produto.
5. Um arquiteto agrega valor real a uma organização não perseguindo bala de prata após bala de prata, mas sim aprimorando suas habilidades em analisar os trade-offs à medida que aparecem.
6. Uma vez que todos aprendemos a isolar dimensões e



realizar a análise de trade-off, estamos aprendendo coisas concretas sobre nossa arquitetura.

7. Encontrar o melhor contexto para uma decisão permite ao arquiteto considerar menos opções, simplificando enormemente o processo decisório.

Capítulo 17 | Frases das páginas 8115-8437

1. Se não fosse pela orientação, os insights e o conhecimento do Logan, não estaríamos na situação em que estamos agora.
2. Continuamos nosso novo hábito de criar tabelas de trade-off para todas as nossas decisões... isso é ARQUITETURA DE SOFTWARE. E como você pode ver, funciona.
3. Não podemos realmente confiar em conselhos genéricos para nossa ARQUITETURA DE SOFTWARE—é muito diferente de todas as outras. Temos que fazer o trabalho duro de análise de trade-off constantemente.
4. Aprendemos a isolar dimensões e realizar análise de trade-off, estamos aprendendo coisas concretas sobre nossa



ARQUITETURA DE SOFTWARE.

5.Quanto mais fatos concretos podemos aprender sobre nosso ecossistema único, mais precisa pode se tornar nossa análise.

Capítulo 18 | Frases das páginas 8438-8441

- 1.Cada decisão da Sysops Squad neste livro foi acompanhada por um correspondente Registro de Decisão de Arquitetura.
- 2.Migração Usando a Abordagem de Decomposição Baseada em Componentes
- 3.Uso de Banco de Dados Documental para Pesquisa de Clientes
- 4.Uso de um Sidecar para Acoplamento Operacional
- 5.Uso de Caching Replicado em Memória para Dados de Perfis de Especialistas



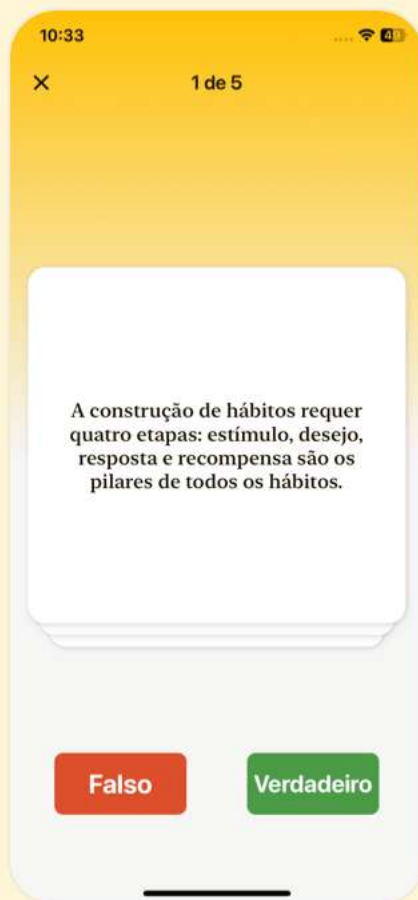


Baixe o app Bookey para desfrutar

Mais de 1000 resumos de livros com quizzes

Teste grátis disponível!

Escanear para baixar



Capítulo 19 | Frases das páginas 8442-8450

- 1.ADR: Migrar a Aplicação do Squad de Sysops para uma ARQUITETURA DE SOFTWARE Distribuída
- 2.ADR: Uso de Banco de Documentos para Pesquisa de Clientes
- 3.ADR: Uso de uma Biblioteca Compartilhada para Lógica Comum da Base de Dados de Chamadas
- 4.ADR: Uso de Orquestração para o Fluxo de Trabalho de Chamadas Primárias
- 5.ADR: Contrato Flexível para Aplicativo Móvel do Squad de Sysops

Capítulo 20 | Frases das páginas 8451-8596

- 1.O principal foco deste livro é a análise de trade-offs; para tal, criamos uma série de tabelas e figuras na Parte II para resumir os trade-offs em torno de uma preocupação arquitetônica específica.
- 2.Trade-offs para a técnica de replicação de código



3.Trade-offs para orquestração

4.Trade-offs da técnica de divisão de propriedade conjunta

5.A importância dos dados na arquitetura

6.Usando Funções de Fitness.

7.A armadilha do ‘Fora de Contexto’

Mais livros gratuitos no Bookey



Escanear para baixar

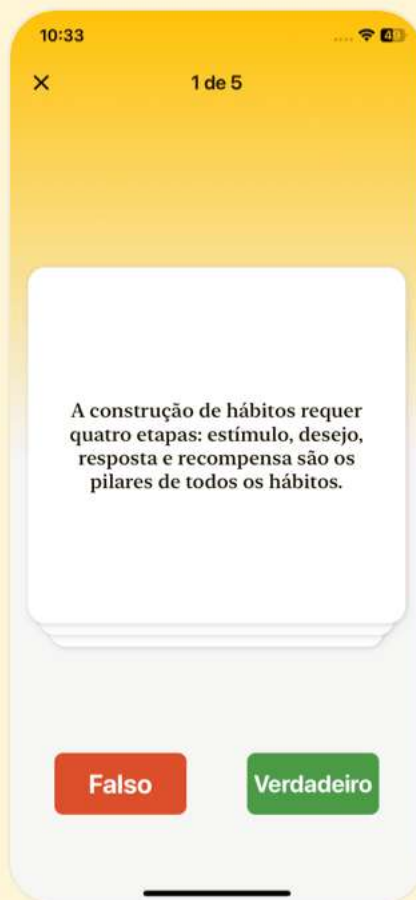
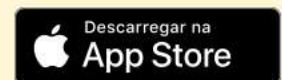


Baixe o app Bookey para desfrutar

Mais de 1000 resumos de livros com quizzes

Teste grátis disponível!

Escanear para baixar



ARQUITETURA DE SOFTWARE

Perguntas

Ver no site do Bookey

Capítulo 1 | 1. O Que Acontece Quando Não Há “Melhores Práticas”? | Perguntas e respostas

1.Pergunta

Qual é o papel dos arquitetos quando as melhores práticas não estão disponíveis?

Resposta: Os arquitetos têm a tarefa de enfrentar desafios únicos e avaliar compromissos em situações novas, uma vez que muitos problemas de arquitetura não possuem soluções estabelecidas.

2.Pergunta

Por que os arquitetos devem evitar buscar soluções milagrosas?

Resposta: Soluções milagrosas são raras; em vez disso, os arquitetos devem aceitar que o melhor design pode ser simplesmente o menor dos males entre fatores concorrentes.

3.Pergunta

O que define as partes 'difíceis' da arquitetura, conforme

Mais livros gratuitos no Bookey



Escanear para baixar

discutido no texto?

Resposta:As partes 'difíceis' da arquitetura referem-se aos problemas complexos que os arquitetos enfrentam e aos elementos estruturais fundamentais que são menos suscetíveis a mudanças ao longo do tempo.

4.Pergunta

Por que a compreensão dos estilos arquitetônicos históricos é importante?

Resposta:Compreender os estilos arquitetônicos históricos ajuda os arquitetos a reconhecer as limitações que moldaram as práticas atuais e os orienta na adaptação às tecnologias em evolução.

5.Pergunta

Quão crucial é o dado na ARQUITETURA DE SOFTWARE?

Resposta:Os dados são vitais, durando mais do que os sistemas e influenciando a ARQUITETURA DE SOFTWARE contemporânea ao exigir que os dados corretos sejam acessíveis e utilizáveis.



6.Pergunta

O que são Registros de Decisão Arquitetônica (ADR)?

Resposta:Os ADRs são registros documentados de decisões arquitetônicas, incluindo contexto, a decisão em si e suas consequências, servindo como referência para considerações de design futuras.

7.Pergunta

O que são funções de aptidão arquitetônica e por que são importantes?

Resposta:Funções de aptidão são mecanismos que avaliam a integridade das características arquitetônicas de forma objetiva, ajudando a reforçar princípios de design e a manter a qualidade ao longo do processo de desenvolvimento.

8.Pergunta

O que significa quando o texto afirma 'os dados tendem a viver muito mais do que os sistemas'?

Resposta:Isso indica que os dados superam as aplicações que os utilizam, exigindo planejamento e design cuidadosos para garantir sua longevidade e adaptabilidade.

9.Pergunta

Mais livros gratuitos no Bookey



Escanear para baixar

Por que o feedback contínuo é enfatizado nas práticas modernas de desenvolvimento de software?

Resposta:O feedback contínuo, possibilitado pela automação, reduz a probabilidade de erros e permite que as equipes de desenvolvimento avaliem rapidamente o impacto das mudanças, aumentando a produtividade geral.

10.Pergunta

Qual é a importância de manter as atividades de arquitetura e design distintas?

Resposta:Manter essa distinção permite que os arquitetos se concentrem em princípios e compromissos gerais, em vez de se perderem em detalhes de implementação.

Capítulo 2 | I. Desmontando as Coisas| Perguntas e respostas

1.Pergunta

Quais são os dois tipos de acoplamento na ARQUITETURA DE SOFTWARE e como eles diferem?

Resposta:Os dois tipos de acoplamento na ARQUITETURA DE SOFTWARE são o acoplamento estático e o acoplamento dinâmico. O



acoplamento estático se refere à forma como as partes arquiteturais (como classes e serviços) estão interligadas e suas dependências, que muitas vezes podem ser medidas em tempo de compilação. Em contraste, o acoplamento dinâmico envolve como essas partes se comunicam em tempo de execução, incluindo o tipo de comunicação e as informações trocadas entre elas.

2.Pergunta

Por que é desafiador para os arquitetos gerenciar a comunicação de serviços em arquiteturas distribuídas?

Resposta:Os arquitetos enfrentam desafios na gestão da comunicação de serviços pois precisam equilibrar as concessões entre desacoplar serviços para garantir independência e manter uma comunicação eficaz. Essa complexidade aumenta com a natureza refinada dos microserviços, onde a orquestração, a transacionalidade e a assíncronicidade apresentam questões significativas.

3.Pergunta



O que significa o conceito de 'quantum arquitetônico' e por que é importante?

Resposta: O quantum arquitetônico significa um artefato independente e implantável definido por alta coesão funcional, alto acoplamento estático e acoplamento dinâmico síncrono. É essencial porque ajuda os arquitetos a entender o escopo, as características de implantação e as dependências dos componentes arquiteturais, orientando a análise de concessões.

4.Pergunta

Como o acoplamento estático afeta as escolhas de design em microserviços?

Resposta: O acoplamento estático impacta as escolhas de design ao definir quais dependências são necessárias para que um serviço funcione. Um alto acoplamento estático geralmente limita a capacidade de os serviços serem implantados de forma independente, uma vez que podem compartilhar componentes cruciais, como bancos de dados.

5.Pergunta



O desacoplamento pode ser prejudicial em arquiteturas distribuídas?

Resposta: Sim, o desacoplamento extremo pode levar a situações em que os componentes não conseguem se comunicar efetivamente, dificultando a construção de sistemas funcionais. Os arquitetos devem encontrar um equilíbrio no acoplamento que permite a comunicação sem comprometer a independência dos serviços.

6.Pergunta

Explique a importância da citação de Paracelso no contexto da ARQUITETURA DE SOFTWARE.

Resposta: A citação 'Todas as coisas são veneno, e nada está isento de veneno; a dosagem sozinha faz com que uma coisa não seja um veneno' destaca que o acoplamento na ARQUITETURA DE SOFTWARE não é inerentemente ruim; como qualquer coisa, depende de quanto e em que contexto é aplicado. Compreender o nível apropriado de acoplamento é crucial para criar sistemas funcionais e manuteníveis.



7.Pergunta

Quais são as dimensões do acoplamento dinâmico quântico e por que são relevantes para os arquitetos?

Resposta:As dimensões do acoplamento dinâmico quântico incluem Comunicação (síncrona vs assíncrona), Consistência (integridade da transação) e Coordenação (gestão do fluxo de trabalho). Estas dimensões orientam os arquitetos na tomada de decisões informadas sobre como os serviços interagem, influenciando o desempenho, a escalabilidade e a gestão de transações.

8.Pergunta

Qual é o papel da orquestração e da coreografia na comunicação de serviços, e como elas diferem?

Resposta:A orquestração refere-se a um orquestrador central gerenciando a comunicação entre os serviços, enquanto a coreografia envolve serviços comunicando-se diretamente entre si sem uma autoridade central de gestão. Os arquitetos devem escolher entre esses padrões com base na complexidade do fluxo de trabalho e nos requisitos de



interação dos serviços.

9.Pergunta

Por que a implantabilidade independente é importante em um quantum arquitetônico?

Resposta:A implantabilidade independente permite que os serviços sejam atualizados e mantidos separadamente, sem impactar outros, aumentando a agilidade e reduzindo o risco de falhas em todo o sistema. É uma característica crítica para microserviços, promovendo práticas eficientes de desenvolvimento e operação.

Capítulo 3 | 2. Discernindo o Acoplamento na ARQUITETURA DE SOFTWARE| Perguntas e respostas

1.Pergunta

Quais são os desafios que os arquitetos enfrentam ao projetar microserviços?

Resposta:Os arquitetos enfrentam dificuldades nas decisões sobre granularidade e comunicação devido à ausência de diretrizes universais claras ou melhores práticas para sistemas complexos. Eles



devem gerenciar orquestração, transacionalidade e assincronismo de uma maneira que equilibre desacoplamento e funcionalidade.

2.Pergunta

Por que o desacoplamento não é inerentemente bom na ARQUITETURA DE SOFTWARE?

Resposta:Embora o desacoplamento seja frequentemente elogiado, sistemas excessivamente desacoplados podem levar a problemas de comunicação onde nada se conecta efetivamente. Os arquitetos devem aplicar o acoplamento de forma apropriada para garantir que os serviços ainda possam funcionar e interagir.

3.Pergunta

O que é acoplamento no contexto da ARQUITETURA DE SOFTWARE?

Resposta:Acoplamento refere-se à dependência entre duas partes de um sistema de software; uma mudança em um componente pode exigir mudanças em outro. Isso pode ser classificado em acoplamento estático (como os serviços estão



conectados) e acoplamento dinâmico (como os serviços se comunicam em tempo de execução).

4.Pergunta

Como o conceito de quantum arquitetural auxilia no design arquitetônico de software?

Resposta:Um quantum arquitetural é uma unidade implantável de forma independente, caracterizada por alta coesão funcional, permitindo um gerenciamento preciso das dependências e comunicação entre os serviços. Isso permite que os arquitetos analisem e otimizem os sistemas para manutenibilidade, escalabilidade e outras qualidades arquitetônicas.

5.Pergunta

Quais são as implicações do acoplamento estático versus dinâmico para microserviços?

Resposta:O acoplamento estático envolve as dependências fundamentais necessárias para a operação, como bancos de dados ou frameworks, enquanto o acoplamento dinâmico afeta como os serviços interagem durante a execução por



meio de protocolos de comunicação. Compreender ambos ajuda os arquitetos a projetar sistemas coesos e flexíveis.

6.Pergunta

Como a modularidade arquitetônica contribui para a agilidade nos negócios?

Resposta:A modularidade arquitetônica melhora a agilidade nos negócios ao permitir que os sistemas sejam divididos em serviços menores e gerenciáveis que podem ser desenvolvidos e implantados de forma independente. Isso possibilita respostas mais rápidas às mudanças do mercado, reduz riscos de implantação e facilita a inovação contínua.

7.Pergunta

O que significa alta coesão funcional na ARQUITETURA DE SOFTWARE?

Resposta:Alta coesão funcional significa que componentes ou funcionalidades relacionadas estão agrupadas de forma próxima, muitas vezes dentro de um único serviço. Isso leva a uma melhor manutenibilidade e torna mais fácil aplicar mudanças sem afetar todo o sistema.



8.Pergunta

Qual é o papel da testabilidade no design arquitetônico?

Resposta:A testabilidade é crucial para a agilidade arquitetônica; ela mede quão facilmente um sistema pode ser testado. Alta testabilidade permite uma validação eficiente das mudanças, levando a uma maior confiança nas implantações. A modularidade arquitetônica geralmente melhora a testabilidade ao isolar mudanças em componentes específicos.

9.Pergunta

Como a analogia do 'copinho de água' explica a modularidade arquitetônica?

Resposta:A analogia do copinho de água ilustra que, à medida que as aplicações crescem, os recursos se tornam sobrecarregados. Separar uma aplicação monolítica em múltiplos 'copos' (serviços) aumenta a capacidade ao permitir que o sistema escale horizontalmente, suportando mais usuários e funcionalidades de forma eficiente.

10.Pergunta

Qual é uma compensação significativa ao fazer a

Mais livros gratuitos no Bookey



Escanear para baixar

transição para uma ARQUITETURA DE MICROSERVIÇOS?

Resposta: Uma grande compensação é a complexidade acrescida na gestão da comunicação entre serviços e das dependências, o que pode impactar negativamente o desempenho e a agilidade se não for controlado com cuidado, resultando em um cenário de 'bola de lama distribuída'.

Mais livros gratuitos no Bookey



Escanear para baixar



Ler, Compartilhar, Empoderar

Conclua Seu Desafio de Leitura, Doe Livros para Crianças Africanas.

O Conceito



Esta atividade de doação de livros está sendo realizada em conjunto com a Books For Africa. Lançamos este projeto porque compartilhamos a mesma crença que a BFA: Para muitas crianças na África, o presente de livros é verdadeiramente um presente de esperança.

A Regra



Ganhe 100 pontos



Resgate um livro



Doe para a África

Seu aprendizado não traz apenas conhecimento, mas também permite que você ganhe pontos para causas beneficentes! Para cada 100 pontos ganhos, um livro será doado para a África.

Teste gratuito com Bookey



Capítulo 4 | 3. Modularidade Arquitetônica|

Perguntas e respostas

1.Pergunta

Qual é a melhor maneira de convencer líderes empresariais a investir em mudanças arquitetônicas?

Resposta: Para convencer efetivamente os líderes empresariais, os arquitetos devem construir um forte argumento comercial que destaque os benefícios da modularidade arquitetônica, correlacione esses benefícios com os problemas enfrentados no sistema atual e delineie claramente os trade-offs envolvidos em um esforço de migração.

2.Pergunta

Por que a modularidade arquitetônica é importante?

Resposta: A modularidade arquitetônica é crucial, pois permite maior escalabilidade, agilidade na resposta a mudanças, melhor manutenção e melhor tolerância a falhas, que são necessárias para as demandas empresariais modernas e avanços tecnológicos.

3.Pergunta

Mais livros gratuitos no Bookey



Escanear para baixar

Quais são as duas principais abordagens para decompôr uma aplicação monolítica?

Resposta:As duas principais abordagens para decompôr uma aplicação monolítica são a decomposição baseada em componentes e o fork tático.

4.Pergunta

O que significa o termo 'fork tático' no contexto de ARQUITETURA DE SOFTWARE?

Resposta:Fork tático refere-se a uma abordagem de reestruturação pragmática onde as equipes clonam o monolito e, em seguida, eliminam código desnecessário em vez de extrair a funcionalidade desejada, permitindo mudanças rápidas sem se preocupar com problemas de acoplamento profundo.

5.Pergunta

Como as métricas de abstração e instabilidade podem ajudar na decomposição de uma aplicação?

Resposta:Métricas como abstração e instabilidade avaliam a estrutura interna de uma base de código. Elas ajudam os



arquitetos a avaliar se uma aplicação pode ser decomposta em componentes ou se uma abordagem de fork tático é necessária para a reestruturação.

6.Pergunta

O que significa se uma base de código está na 'zona de inutilidade'?

Resposta:Se uma base de código está na 'zona de inutilidade', isso indica que o código é muito abstrato e difícil de usar, sugerindo um desequilíbrio em sua estrutura que compromete sua eficácia.

7.Pergunta

Qual é uma possível desvantagem da abordagem de fork tático?

Resposta:Uma desvantagem significativa da abordagem de fork tático é que pode resultar em múltiplos serviços contendo código duplicado e caótico, complicando modificações futuras e esforços de manutenção.

8.Pergunta

Como você determina se uma base de código é adequada para decomposição?



Resposta: Para determinar se uma base de código é adequada para decomposição, os arquitetos consideram a estrutura interna analisando métricas de acoplamento, funcionalidade compartilhada e a presença de componentes bem definidos.

9. Pergunta

Quais benefícios a decomposição baseada em componentes oferece em comparação ao fork tático?

Resposta: A decomposição baseada em componentes permite um processo de migração mais controlado e metódico, reduzindo o risco de duplicação de código e garantindo uma base de código mais organizada e de fácil manutenção.

10. Pergunta

Por que é importante manter a discussão sobre arquitetura do projeto relevante para as partes interessadas do negócio?

Resposta: Manter a relevância nas discussões arquitetônicas com as partes interessadas do negócio garante que as decisões estejam alinhadas com as metas empresariais, facilitando a aprovação do orçamento e o engajamento das partes interessadas no processo de migração.



Capítulo 5 | 4. Decomposição Arquitetônica|

Perguntas e respostas

1.Pergunta

Qual é o primeiro passo para desmembrar uma aplicação monolítica de acordo com o padrão Identificar e Dimensionar Componentes?

Resposta:O primeiro passo é identificar e catalogar os componentes arquiteturais da aplicação, em seguida, dimensionar corretamente esses componentes com base no total de declarações que contêm.

2.Pergunta

Quais métricas podem ser usadas para determinar o tamanho de um componente?

Resposta:O número total de declarações dentro de um componente (soma das declarações em todos os arquivos de origem) é uma métrica útil, juntamente com a porcentagem de código que cada componente representa em relação ao código total.

3.Pergunta

Mais livros gratuitos no Bookey



Escanear para baixar

Por que é importante dimensionar corretamente os componentes?

Resposta: O dimensionamento adequado dos componentes ajuda a garantir que eles não sejam muito grandes (o que complica a separação em serviços) nem muito pequenos (o que pode diminuir a modularidade). Idealmente, os componentes devem estar dentro de um a dois desvios padrão do tamanho médio.

4.Pergunta

Quais problemas surgem quando componentes são construídos sobre outros componentes?

Resposta: Quando os componentes são construídos um sobre o outro, eles perdem sua identidade distinta e complicam o processo de formação de serviços. Isso pode levar a dependências de serviços que dificultam a gestão e a manutenção da aplicação.

5.Pergunta

Qual é o resultado da aplicação do padrão Agrupar Componentes de Domínio Comuns?



Resposta:O resultado é a consolidação da lógica de negócio comum em um único componente, reduzindo a redundância no código e potencialmente minimizando o número de serviços que precisam ser criados.

6.Pergunta

Qual é a importância de monitorar as dependências de entrada e saída dos componentes?

Resposta:Monitorar essas dependências ajuda a avaliar a viabilidade de desmembrar uma aplicação monolítica, bem como a estimar o nível geral de esforço necessário para a migração.

7.Pergunta

Como o padrão Componentes Planos ajuda na gestão de componentes?

Resposta:O padrão Componentes Planos garante que os componentes sejam representados apenas como nós folha em uma estrutura de diretório, eliminando classes órfãs e mantendo definições claras de componentes.

8.Pergunta

Qual é o perigo de ter muito código compartilhado ao



desmembrar uma aplicação monolítica?

Resposta: Ter muito código compartilhado pode criar desafios de manutenção, pois resulta em acoplamento forte e potenciais dificuldades em aplicar mudanças em vários serviços ou bibliotecas compartilhadas.

9.Pergunta

Qual é o resultado da aplicação bem-sucedida do padrão Criar Serviços de Domínio?

Resposta: A aplicação bem-sucedida resulta na identificação e extração de domínios de componentes em serviços de domínio implantados separadamente, levando a uma arquitetura baseada em serviços.

10.Pergunta

Por que é crucial criar um diagrama de domínio durante o processo de agrupamento de componentes?

Resposta: Criar um diagrama de domínio ajuda a validar os candidatos a domínio identificados, ilustra os relacionamentos entre os componentes e enfatiza a necessidade de colaboração das partes interessadas no



esforço de migração.

Capítulo 6 | 5. Padrões de Decomposição Baseados em Componentes| Perguntas e respostas

1.Pergunta

Qual é a importância de identificar e dimensionar componentes em uma aplicação monolítica durante a migração?

Resposta:Identificar e dimensionar componentes é crucial porque ajuda os arquitetos a entender quais componentes são muito grandes (resultando em alta colaboração) ou muito pequenos (menos relevantes funcionalmente). Esse conhecimento permite dividir componentes em serviços mais gerenciáveis e modulares, facilitando transições mais suaves para uma ARQUITETURA DE SOFTWARE distribuída, aprimorando, em última análise, a manutenção e escalabilidade.

2.Pergunta

Por que o componente de Relatórios no aplicativo Sysops Squad é considerado muito grande e qual foi a solução

Mais livros gratuitos no Bookey



Escanear para baixar

proposta?

Resposta:O componente de Relatórios representava 33% do código total, tornando-se significativamente maior do que outros componentes. Para resolver isso, Addison decidiu dividi-lo em quatro componentes separados (Relatórios Compartilhados, Relatórios de Tickets, Relatórios de Especialistas, Relatórios Financeiros) para garantir uma distribuição mais equilibrada de funcionalidade e complexidade.

3.Pergunta

Como a história da arquitetura criada para o componente de Relatórios contribuiu para o processo de refatoração?

Resposta:A história da arquitetura delineou a necessidade de refatorar o componente de Relatórios em componentes funcionais menores. Essa abordagem estruturada esclareceu os objetivos da refatoração, permitindo que a equipe de desenvolvimento implementasse as mudanças de forma sistemática, sem perder de vista a funcionalidade do negócio.

4.Pergunta

Mais livros gratuitos no Bookey



Escanear para baixar

Qual é o papel do padrão Gather Common Domain Components na migração de sistemas monolíticos para sistemas distribuídos?

Resposta:Esse padrão ajuda a identificar e consolidar a lógica de negócios sobreposta entre componentes em serviços únicos, minimizando a duplicação de código e simplificando a ARQUITETURA DE SOFTWARE. Essa ação promove eficiência e clareza na ARQUITETURA DE SOFTWARE distribuída resultante, estabelecendo limites de domínio claros.

5.Pergunta

Quais funções de fitness automatizadas podem ajudar a manter o tamanho dos componentes durante o desenvolvimento contínuo da aplicação?

Resposta:Funções de fitness automatizadas, como manter um inventário de componentes, impor porcentagens de tamanho para componentes e verificar distribuições normais dos tamanhos de componentes, ajudam a garantir que nenhum componente se torne um gargalo de desempenho durante o



processo de migração e manutenção.

6.Pergunta

Quais são as vantagens de achatar componentes, conforme descrito no padrão Flatten Components?

Resposta:Achatar componentes assegura que todas as classes estejam organizadas de maneira ordenada dentro de nós folhas (áreas de funcionalidade específica) e evita problemas de classes órfãs acumuladas sob namespaces raiz. Essa estrutura aprimora a clareza, facilita a formação de serviços e, em última análise, melhora a modularidade e a manutenibilidade da aplicação.

7.Pergunta

Por que entender as dependências de componentes é vital antes de desmembrar uma aplicação monolítica?

Resposta:Entender as dependências de componentes é fundamental porque ajuda os arquitetos a avaliar a viabilidade dos esforços de migração, avaliar o nível de acoplamento, prever a complexidade da ARQUITETURA DE SOFTWARE resultante e identificar oportunidades de



refatoração - todos críticos para uma migração bem-sucedida para um modelo distribuído.

8.Pergunta

O que pode ser inferido de um diagrama de dependência que mostra várias dependências entre componentes?

Resposta:Um diagrama de dependência com várias dependências indica alto acoplamento de componentes, sugerindo que desmembrar a aplicação monolítica provavelmente será altamente complexo e pode exigir uma reescrita significativa de código, em vez de apenas refatoração.

9.Pergunta

Qual é a importância das funções de fitness ligadas aos domínios de componentes após a migração?

Resposta:Essas funções de fitness impõem regras estruturais que mantêm a integridade dos domínios de componentes, garantindo que a ARQUITETURA DE SOFTWARE permaneça limpa e gerenciável. Elas previnem a fragmentação e garantem que os componentes permaneçam



alinhados com seus respectivos domínios à medida que a aplicação evolui.

Mais livros gratuitos no Bookey



Escanear para baixar



As melhores ideias do mundo desbloqueiam seu potencial

Essai gratuit avec Bookey



Escanear para baixar



Capítulo 7 | 6. Desmembrando Dados Operacionais| Perguntas e respostas

1.Pergunta

Qual é o principal desafio ao separar um banco de dados monolítico?

Resposta:Separar um banco de dados monolítico é difícil porque os dados estão geralmente altamente acoplados à funcionalidade do aplicativo. Isso significa que identificar limites claros para a separação é complexo, e quaisquer mudanças podem interferir em vários serviços.

2.Pergunta

Como os arquitetos podem justificar um esforço de decomposição de dados?

Resposta:Os arquitetos podem justificar a decomposição de dados ao analisar desintegradores de dados (fatores que incentivam a separação dos dados) e integradores de dados (fatores que mantêm os dados juntos), buscando um equilíbrio entre os dois.

3.Pergunta

Mais livros gratuitos no Bookey



Escanear para baixar

Quais são alguns dos principais fatores de desintegração de dados mencionados?

Resposta: Os principais fatores de desintegração de dados incluem controle de mudanças, gerenciamento de conexões, escalabilidade, tolerância a falhas, considerações de quantum arquitetônico e otimização do tipo de banco de dados.

4.Pergunta

Por que o controle de mudanças é uma preocupação significativa na gestão de bancos de dados?

Resposta: O controle de mudanças é significativo porque alterar esquemas de bancos de dados (por exemplo, excluir ou renomear tabelas) pode afetar vários serviços que dependem desses dados, tornando as atualizações coordenadas difíceis e propensas a erros.

5.Pergunta

Como um contexto delimitado ajuda no controle de mudanças na gestão de bancos de dados?

Resposta: Um contexto delimitado ajuda a isolar mudanças, garantindo que modificações nos dados dentro de um



contexto não impactem outros serviços que utilizam contextos delimitados diferentes, simplificando assim a gestão e reduzindo riscos.

6.Pergunta

Qual é o objetivo de usar um limite de conexões em uma ARQUITETURA DE SOFTWARE distribuída?

Resposta:O uso de um limite de conexões restringe o número de conexões de banco de dados que cada serviço pode utilizar, evitando saturação e garantindo que todos os serviços tenham acesso adequado aos recursos do banco de dados.

7.Pergunta

O que significa escalabilidade no contexto da gestão de bancos de dados?

Resposta:Escalabilidade refere-se à capacidade do banco de dados de lidar com aumentos de carga e demandas de conexão de serviços distribuídos, mantendo o desempenho e os tempos de resposta.

8.Pergunta

Como a separação de um banco de dados aumenta a

Mais livros gratuitos no Bookey



Escanear para baixar

tolerância a falhas?

Resposta: Ao separar um banco de dados, você elimina o único ponto de falha, garantindo que, se um banco de dados ficar fora do ar, apenas os serviços que dependem dele sejam impactados, enquanto outros continuam a funcionar.

9.Pergunta

Qual é a importância do quantum arquitetônico na separação de bancos de dados?

Resposta: Os conceitos de quantum arquitetônico ajudam a determinar quando quebrar o banco de dados; garantindo que cada serviço possa evoluir de forma independente e mantendo alta coesão dentro dos grupos de serviços.

10.Pergunta

Por que é importante selecionar o tipo de banco de dados apropriado para diferentes dados?

Resposta: Selecionar o tipo certo de banco de dados otimiza o desempenho e a eficiência para várias cargas de trabalho de dados, permitindo que o sistema atenda melhor a casos de uso específicos e melhore a funcionalidade geral do



aplicativo.

Capítulo 8 | 7. Granularidade de Serviço| Perguntas e respostas

1.Pergunta

Qual é um dos principais desafios ao determinar a granularidade dos serviços na ARQUITETURA DE SOFTWARE de microserviços?

Resposta:Um dos principais desafios na determinação da granularidade dos serviços é equilibrar as necessidades conflitantes de desintegração (quebrar serviços em partes menores) e integração (combinar serviços para melhor desempenho e confiabilidade). Isso requer uma análise cuidadosa das compensações envolvidas, já que os desenvolvedores frequentemente têm opiniões diferentes sobre o tamanho e o escopo ideais de cada serviço.

2.Pergunta

Como os desintegradores de granularidade guiam o processo de tomada de decisão na separação de serviços?

Mais livros gratuitos no Bookey



Escanear para baixar

Resposta: Os desintegradores de granularidade fornecem razões específicas para quando quebrar um serviço em outros menores. Os principais fatores incluem: escopo e função do serviço, volatilidade do código, necessidades de escalabilidade e throughput, tolerância a falhas, requisitos de segurança e a necessidade de extensibilidade para funcionalidades futuras. Avaliando esses fatores, uma equipe pode justificar a separação dos serviços.

3. Pergunta

Quais métricas podem ser usadas para medir a granularidade dos serviços?

Resposta: Métricas como o número de declarações em um serviço e o número de interfaces ou operações públicas expostas pelo serviço podem medir a granularidade de forma objetiva. Essas métricas ajudam a avaliar a funcionalidade e a coesão de um serviço, indicando se ele deve ser separado ou combinado.

4. Pergunta

Qual é o papel da funcionalidade do cliente na discussão sobre granularidade dos serviços?



Resposta:A funcionalidade do cliente serve como um exemplo crucial nas discussões sobre a granularidade dos serviços. As equipes devem avaliar se aspectos como registro de clientes, faturamento e gerenciamento de perfil devem ser consolidados em um único serviço ou distribuídos entre vários serviços com base em fatores como segurança, frequência de mudanças e coesão.

5.Pergunta

Como a volatilidade do código pode influenciar a decisão de desintegrar um serviço?

Resposta:A volatilidade do código, ou a frequência das mudanças de código, pode justificar a separação de um serviço. Se uma função muda frequentemente enquanto outras permanecem estáveis, separá-la em seu próprio serviço pode reduzir o escopo de testes e o risco de implantação ao fazer atualizações, minimizando o impacto em funcionalidades mais estáveis.

6.Pergunta

Qual é a importância da tolerância a falhas ao avaliar a granularidade dos serviços?



Resposta: A tolerância a falhas é essencial, pois descreve a capacidade de um serviço de continuar operando apesar de falhas. Serviços divididos em componentes menores podem melhorar a tolerância a falhas, isolando erros, garantindo que uma falha não derrube funcionalidades interconectadas.

7. Pergunta

Qual é o princípio da responsabilidade única e como ele se relaciona aos microserviços?

Resposta: O princípio da responsabilidade única postula que cada serviço ou classe deve focar exclusivamente em uma função ou responsabilidade. Na ARQUITETURA DE SOFTWARE de microserviços, esse princípio orienta decisões para criar serviços de único propósito, promovendo um código focado e de fácil manutenção que permite atualizações e expansões mais simples.

8. Pergunta

Por que uma equipe pode decidir manter os serviços consolidados em vez de desintegrá-los?

Resposta: As equipes podem optar por manter os serviços



juntos por várias razões, incluindo a necessidade de transações ACID para garantir a integridade dos dados entre funções relacionadas, a eficiência de reduzir a comunicação entre serviços, o que pode eliminar latências e a manutenção da confiabilidade e consistência dos dados.

9.Pergunta

Em quais situações um proprietário de produto ou patrocinador do negócio deve estar envolvido nas decisões sobre a granularidade dos serviços?

Resposta:Um proprietário de produto ou patrocinador do negócio deve estar envolvido em situações onde as compensações entre agilidade (tempo de colocação no mercado) e necessidades críticas do negócio, como integridade dos dados, segurança e desempenho, precisam ser negociadas. Sua contribuição ajuda a alinhar decisões técnicas com os objetivos de negócios, garantindo que as decisões sejam práticas e viáveis.

10.Pergunta

Você pode ilustrar o conceito de extensibilidade com um exemplo do texto?



Resposta:A extensibilidade pode ser ilustrada pelo contexto de um serviço de pagamento que precisa acomodar diferentes métodos de pagamento, como cartões de crédito, cartões-presente e outros. Em vez de manter um serviço monolítico onde adicionar um novo tipo de pagamento exigiria retestar o serviço inteiro, dividi-lo em serviços menores e dedicados (como Processamento de Cartão de Crédito e Processamento de Cartão-Presente) pode simplificar a adição de novas opções de pagamento, melhorando assim a capacidade de resposta e agilidade.

Capítulo 9 | II. Reunindo as peças| Perguntas e respostas

1.Pergunta

Quais são os principais desafios ao separar um módulo coeso em uma ARQUITETURA DE SOFTWARE?

Resposta:Os principais desafios incluem o aumento do acoplamento e a diminuição da legibilidade, tornando necessário que os arquitetos pensem sobre como as partes estruturais interagem entre si em vez de se concentrar apenas em suas funcionalidades



individuais. Os arquitetos também devem lidar com trade-offs relacionados à comunicação entre serviços, contratos, transações distribuídas e acesso a dados.

2.Pergunta

Por que a reutilização de código é considerada uma parte difícil da arquitetura distribuída?

Resposta:A reutilização de código é desafiadora em arquiteturas distribuídas devido à complexidade de gerenciar funcionalidades compartilhadas entre múltiplos serviços. Isso inclui questões sobre como compartilhar código efetivamente, se deve-se replicar código entre serviços e como gerenciar versionamento e dependências sem criar uma rede emaranhada de serviços acoplados.

3.Pergunta

Quais são os trade-offs associados ao uso de bibliotecas compartilhadas na ARQUITETURA DE SOFTWARE?

Resposta:As vantagens das bibliotecas compartilhadas incluem a capacidade de versionar mudanças, reduzir erros



em tempo de execução e permitir boa agilidade para mudanças de código. As desvantagens incluem dificuldades em gerenciar dependências, potencial duplicação de código em bases de código heterogêneas e desafios na comunicação de versões.

4.Pergunta

Como a técnica de serviço compartilhado difere da técnica de biblioteca compartilhada?

Resposta:A técnica de serviço compartilhado permite que funcionalidades comuns sejam fornecidas como um serviço separado acessado em tempo de execução, o que pode simplificar a implantação, mas introduz riscos associados a mudanças em tempo de execução. A técnica de biblioteca compartilhada vincula o código compartilhado em tempo de compilação, reduzindo o risco em tempo de execução, mas complicando o gerenciamento de dependências.

5.Pergunta

Qual é o papel do padrão Sidecar na ARQUITETURA DE MICROSERVIÇOS?



Resposta:O padrão Sidecar ajuda a desacoplar capacidades operacionais, como logging, monitoramento e autenticação, da lógica de domínio ao colocar essas funcionalidades em um componente separado. Isso permite um comportamento operacional consistente entre os serviços, ao mesmo tempo que reduz a complexidade que cada serviço deve gerenciar de forma independente.

6.Pergunta

Quando é apropriado usar replicação de código e quais são suas vantagens?

Resposta:A replicação de código é apropriada para código estático simples ou classes pontuais que provavelmente não mudarão. As vantagens incluem manter um contexto limitado e evitar o compartilhamento de código, mas isso apresenta riscos quando atualizações ou correções de bugs são necessárias, pois elas devem ser propagadas por todas as instâncias replicadas.

7.Pergunta

Qual é a importância do versionamento em bibliotecas compartilhadas e quais complexidades surgem com isso?



Resposta:O versionamento permite que os serviços adotem mudanças da biblioteca compartilhada sem exigir que todos os consumidores atualizem simultaneamente. No entanto, isso introduz complexidades, como a comunicação de mudanças de versão entre equipes e a gestão de estratégias de depreciação para versões mais antigas da biblioteca.

8.Pergunta

O que os arquitetos devem considerar ao decidir entre serviços compartilhados e bibliotecas compartilhadas?

Resposta:Os arquitetos devem considerar trade-offs como risco de mudança, implicações de desempenho, escalabilidade e tolerância a falhas. Os serviços compartilhados permitem atualizações dinâmicas, mas podem introduzir erros em tempo de execução, enquanto as bibliotecas compartilhadas oferecem estabilidade em tempo de compilação, mas podem complicar o gerenciamento de dependências.

9.Pergunta

Quais são as capacidades operacionais que devem ser gerenciadas por meio de um Sidecar, de acordo com o



texto?

Resposta:As capacidades operacionais que devem ser gerenciadas por meio de um Sidecar incluem monitoramento, logging, descoberta de serviços, autenticação e autorização. Isso ajuda a garantir consistência entre os serviços e alivia a pressão sobre equipes individuais para gerenciar essas capacidades de forma independente.

10.Pergunta

Quais lições podem ser retiradas das ARQUITETURAS de organizações que priorizam a reutilização?

Resposta:Os alvos de reutilização simples, como sistemas operacionais e frameworks estáveis, são eficazes porque têm taxas de mudança bem compreendidas. Em contraste, o acoplamento de capacidades internas que mudam rapidamente pode levar à fragilidade. Portanto, os arquitetos devem garantir que a reutilização seja informada tanto por estratégias de abstração quanto pela taxa de mudança antecipada para componentes compartilhados.



Ad



Escanear para baixar



Experimente o aplicativo Bookey para ler mais de 1000 resumos dos melhores livros do mundo

Desbloqueie **1000+** títulos, **80+** tópicos

Novos títulos adicionados toda semana

Product & Brand

Liderança & Colaboração

Gerenciamento de Tempo

Relacionamento & Comunicação

Estratégia de Negócios

Criatividade

Memórias

Conheça a Si Mesmo

Psicologia Positiva

Empreendedorismo

História Mundial

Comunicação entre Pais e Filhos

Autocuidado

Mindfulness

Visões dos melhores livros do mundo

Desenvolvimento pessoal

Os 7 Hábitos das Pessoas Altamente Eficazes



Mini Hábitos



Hábitos Atômicos



O Clube das 5 da Manhã



Como Fazer Amigos e Influenciar Pessoas



Como Não



Teste gratuito com Bookey



Capítulo 10 | 8. Padrões de Reuso| Perguntas e respostas

1.Pergunta

Quais são os principais trade-offs entre usar uma biblioteca compartilhada e um serviço compartilhado em uma ARQUITETURA DE SOFTWARE distribuída?

Resposta:Os principais trade-offs entre usar uma biblioteca compartilhada e um serviço compartilhado incluem:

1. ****Controle de Mudanças vs. Gerenciamento de Dependências****: Bibliotecas compartilhadas geralmente resultam em um gerenciamento de dependências mais fácil quando todos os serviços usam uma biblioteca comum, mas mudar uma biblioteca pode exigir retestes e novas implantações de todos os serviços que a utilizam. Serviços compartilhados permitem atualizações mais granulares, mas introduzem complexidades na gestão de versões de API e garantia de compatibilidade entre várias dependências.



2. ****Mudanças em Tempo de Execução vs. Tempo de Compilação****: Mudanças em bibliotecas compartilhadas são tipicamente em tempo de compilação, minimizando falhas em tempo de execução, enquanto mudanças em serviços compartilhados ocorrem em tempo de execução, o que pode resultar em problemas de compatibilidade ou falhas em serviços que dependem dessas mudanças.

3. ****Impacto na Performance****: Chamar serviços compartilhados implica em latência de rede devido à comunicação entre serviços, enquanto bibliotecas compartilhadas não têm essa latência, pois estão compiladas no próprio serviço, resultando em uma performance mais rápida.

4. ****Preocupações com Escalabilidade****: Serviços compartilhados precisam escalar de forma independente com base no uso de múltiplos serviços dependentes, o que pode complicar a



ARQUITETURA, enquanto bibliotecas compartilhadas escalam naturalmente com os serviços que as incluem.

5. ****Tolerância a Falhas****: Se um serviço compartilhado falhar ou se tornar indisponível, todos os serviços dependentes também podem falhar, enquanto uma falha na funcionalidade de uma biblioteca compartilhada fica isolada ao serviço que a utiliza.

6. ****Complexidade de Versionamento****: Bibliotecas compartilhadas podem usar versionamento para permitir que diferentes serviços adotem atualizações em seu próprio ritmo; serviços compartilhados exigem gerenciar versões de API em meio a interdependências potencialmente complexas.

2.Pergunta

Quando pode ser vantajoso usar replicação de código em uma ARQUITETURA DE SOFTWARE distribuída?

Resposta:A replicação de código pode ser vantajosa em



cenários que envolvem código estático e estável que provavelmente não precisará de mudanças ou atualizações frequentes. Isso pode incluir cenários onde:

1. ****Código Único e Específico****: O código que está sendo replicado não precisa ser reutilizado entre serviços ou é específico para um único serviço (por exemplo, funções utilitárias únicas).
2. ****Simplicidade****: Para sistemas legados migrando para microserviços, classes de utilitário estáticas simples ou anotações de configuração que não mudam frequentemente podem ser replicadas sem a sobrecarga de configurar bibliotecas compartilhadas.
3. ****Complexidade Reduzida****: A replicação de código pode simplificar a complexidade da ARQUITETURA, evitando dependências compartilhadas, tornando mais fácil manter contextos limitados.
4. ****Necessidades de Performance****: Em certas situações, ter código diretamente disponível pode melhorar a performance ao eliminar a necessidade de chamadas



inter-serviços adicionais para bibliotecas ou serviços compartilhados.

3.Pergunta

Quais são alguns riscos associados à adoção de um único DLL compartilhado em uma ARQUITETURA DE SOFTWARE distribuída?

Resposta:A adoção de um único DLL compartilhado apresenta vários riscos:

1. ****Fragilidade****: Atualizações no DLL compartilhado podem repercutir em todos os serviços que o utilizam, exigindo testes e implantações coordenadas para garantir compatibilidade, o que leva a um aumento na sobrecarga de manutenção.
2. ****Problemas de Acoplamento****: Um único DLL compartilhado pode criar um acoplamento forte entre serviços, onde uma mudança no DLL pode quebrar inadvertidamente múltiplos serviços se não forem testados adequadamente.
3. ****Dificuldade na Gestão de Versões****: Manter o controle



de versão se torna desafiador porque todos os serviços devem adotar a mesma versão do DLL, aumentando o risco caso uma mudança quebradora ocorra.

4. ****Riscos de Regressão****: Se um defeito for introduzido no DLL, pode afetar todos os serviços que dependem dele, aumentando assim a chance de problemas generalizados em todo o sistema.

5. ****Escalabilidade Limitada****: Mudanças no DLL compartilhado podem necessitar que todos os serviços dependentes escalem juntos, complicando a gestão de recursos.

4.Pergunta

Por que o versionamento é importante para bibliotecas compartilhadas e que desafios ele introduz?

Resposta:O versionamento é crucial para bibliotecas compartilhadas porque:

1. ****Garante Compatibilidade Reversa****: Permite que os serviços adotem mudanças sem serem forçados a atualizar todos de uma só vez, proporcionando flexibilidade e



minimizando o tempo de inatividade.

2. ****Facilita Mudanças Seguras****: As equipes podem introduzir novas versões da biblioteca que contêm funcionalidades adicionais ou correções de bugs enquanto deixam versões antigas disponíveis para serviços que não podem atualizar rapidamente.

No entanto, o versionamento introduz desafios como:

1. ****Comunicação Complexa****: Em um sistema distribuído com diversas equipes, informar todas as partes relevantes sobre mudanças de versão e seus impactos pode ser complexo.
2. ****Rastreamento e Manutenção de Versões****: Garantir que as equipes utilizem a versão correta e gerenciar versões obsoletas pode ser trabalhoso, especialmente em sistemas grandes com muitas dependências.
3. ****Complexidade nos Testes****: Com múltiplas versões em uso simultaneamente, testar todas as interações potenciais se torna mais complexo, aumentando a coordenação necessária para atualizações.



5.Pergunta

Qual é o conceito de acoplamento operacional e como ele influencia decisões de design na ARQUITETURA DE SOFTWARE de microserviços?

Resposta:O acoplamento operacional refere-se a cenários onde diferentes microserviços compartilham preocupações operacionais (por exemplo, logging, monitoramento, descoberta de serviços) enquanto mantêm lógica de domínio separada. Isso influencia decisões de design ao:

1. ****Incentivar a Duplicação em vez do Acoplamento****:

Promovendo a ideia de que duplicar capacidades operacionais comuns entre serviços pode ser melhor do que criar dependências que podem levar a acoplamento mais forte e fragilidade.

2. ****Promover Padrões de Sidecar e Service Mesh****: Forma a justificativa para padrões arquiteturais como sidecars que desacoplam geograficamente tarefas operacionais da funcionalidade de domínio, permitindo consistência sem acoplar os serviços fortemente.



3. ****Avaliar Riscos de Acoplamento****: Influencia como os serviços interagem, incentivando o design de interfaces mais limpas e separação de preocupações que preserve a independência do ciclo de vida e evolução de cada microserviço.

6.Pergunta

Como o padrão Sidecar facilita a ARQUITETURA DE SOFTWARE de microserviços?

Resposta:O padrão Sidecar facilita a ARQUITETURA DE SOFTWARE de microserviços ao:

1. ****Desacoplar Responsabilidades Operacionais****: Permite que preocupações operacionais (como logging, monitoramento e segurança) sejam gerenciadas no componente sidecar sem afetar a lógica de negócios central de cada microserviço.
2. ****Criar Interfaces Consistentes****: Garantindo que as capacidades operacionais sejam integradas de maneira uniforme em todos os serviços através do sidecar, o que evita o caos de implementação causado por escolhas de soluções



dísparas feitas por diferentes equipes.

3. ****Facilitar Manutenção e Atualizações****: Fornece um ponto centralizado para gerenciar atualizações nas capacidades operacionais sem a necessidade de reimplementar cada serviço individualmente, melhorando a agilidade e reduzindo o tempo de inatividade.

4. ****Consolidar Funções Complexas****: Operações como descoberta de serviços, gerenciamento de APIs ou circuit breaking podem ser implementadas uma vez no sidecar, em vez de serem duplicadas em múltiplos serviços, aumentando a produtividade dos desenvolvedores.

7.Pergunta

O que é acoplamento ortogonal e por que é significativo no contexto de microserviços?

Resposta:O acoplamento ortogonal na ARQUITETURA DE SOFTWARE de microserviços refere-se à independência de dois componentes ou preocupações que ainda precisam se intersectar para formar uma solução completa. Isso é significativo porque:



1. ****Promove Independência****: Permite que diferentes componentes da ARQUITETURA (como lógica operacional e de domínio) evoluam de forma independente, reduzindo o risco de que mudanças em uma área impactem outra.
2. ****Minimiza Complexidade****: Ao definir limites claros entre preocupações—como separar loops de interface de usuário do desenvolvimento de lógica de negócios—os arquitetos podem gerenciar dependências de maneira mais eficaz, levando a sistemas simplificados que são mais fáceis de entender e manter.
3. ****Aumenta a Flexibilidade****: Reconhecer o acoplamento ortogonal dá às equipes a liberdade de implementar upgrades ou mudanças específicas para seus domínios sem exigir interação entre domínios cruzados, permitindo assim ciclos de inovação mais rápidos enquanto mantém a estabilidade do sistema.

Capítulo 11 | 9. Propriedade de Dados e Transações Distribuídas| Perguntas e respostas

1.Pergunta

Qual é a regra básica para atribuir a propriedade de



tabelas em uma ARQUITETURA DE SOFTWARE distribuída?

Resposta:O serviço que realiza operações de escrita em uma tabela é o proprietário dessa tabela.

2.Pergunta

Por que a propriedade conjunta de tabelas pode complicar a ARQUITETURA DE DADOS?

Resposta:A propriedade conjunta pode complicar a ARQUITETURA DE DADOS porque envolve múltiplos serviços escrevendo na mesma tabela, tornando desafiador determinar qual serviço deve ser o proprietário dos dados.

3.Pergunta

Quais são algumas desvantagens da propriedade comum de uma tabela entre vários serviços?

Resposta:A propriedade comum leva a problemas como desafios de escalabilidade, tolerância a falhas e governança complexa, já que vários serviços devem concordar sobre como gerenciar mudanças de dados e acesso.

4.Pergunta

Qual é uma possível solução para a propriedade conjunta



se dividir a tabela não for viável?

Resposta: Usar a técnica de delegado, onde um serviço é designado como o único proprietário da tabela, e outros se comunicam com esse serviço para realizar atualizações.

5.Pergunta

Como os padrões de sincronização em segundo plano impactam o contexto delimitado em sistemas distribuídos?

Resposta: Os padrões de sincronização em segundo plano podem quebrar contextos delimitados ao acoplar fontes de dados, o que pode levar a complexidade na gestão de dados e aumentar o risco de inconsistências.

6.Pergunta

O que é BASE e como ele difere de ACID?

Resposta: BASE significa Disponibilidade Básica, Estado Soft e Consistência Eventual, que se aplica a transações distribuídas. Ao contrário do ACID, que garante atomicidade, consistência, isolamento e durabilidade em uma única transação, BASE foca na disponibilidade e permite



inconsistências temporárias.

7.Pergunta

Por que a consolidação de serviços é uma espada de dois gumes em termos de gestão de dados e ARQUITETURA DE SISTEMAS?

Resposta:Embora a consolidação de serviços possa resolver dependências e melhorar o desempenho ao ter uma estrutura de propriedade única, ela aumenta o escopo de testes, o risco de implantação e pode levar a uma redução na tolerância a falhas, já que todas as partes do serviço devem escalar juntas.

8.Pergunta

Como o padrão baseado em eventos pode ser vantajoso para alcançar consistência eventual em uma ARQUITETURA DE SOFTWARE distribuída?

Resposta:O padrão baseado em eventos permite que os serviços sejam altamente desacoplados e responsivos, alcançando consistência de dados em tempo hábil por meio do manuseio assíncrono de eventos.

9.Pergunta

Qual é o impacto de um serviço orquestrador na



responsividade e consistência durante transações distribuídas?

Resposta: Usar um serviço orquestrador geralmente favorece a consistência em detrimento da responsividade, pois requer mais comunicação em rede e pode introduzir latências dependendo de como as solicitações são gerenciadas.

10.Pergunta

Quais são as consequências de usar a técnica de delegado para a propriedade de dados em sistemas distribuídos?

Resposta: A técnica de delegado pode criar um alto nível de acoplamento entre os serviços, complicar o desempenho para operações de escrita por serviços não proprietários e pode levar a problemas de tolerância a falhas e falta de transações atômicas.

Capítulo 12 | 10. Acesso a Dados Distribuídos| Perguntas e respostas

1.Pergunta

Quais são os principais desafios em arquiteturas distribuídas em relação ao acesso a dados?

Resposta: Em arquiteturas distribuídas, os



principais desafios relacionados ao acesso a dados incluem lidar com a latência devido a chamadas de rede, garantir escalabilidade sem dependências de serviços, gerenciar a consistência e a propriedade dos dados, e enfrentar possíveis problemas de desempenho causados pela comunicação entre serviços.

2.Pergunta

Por que é difícil acessar dados pertencentes a outro serviço em uma arquitetura de microserviços?

Resposta:Acessar dados pertencentes a outro serviço em uma arquitetura de microserviços é difícil devido a limites mais rigorosos e à separação de responsabilidades entre os serviços. Cada serviço pode manter seu próprio banco de dados, o que cria desafios para a recuperação de dados, especialmente se a comunicação síncrona for necessária; a latência da rede pode atrasar significativamente as respostas.

3.Pergunta

O que é o padrão de Comunicação entre Serviços e quais são suas desvantagens?



Resposta:O padrão de Comunicação entre Serviços envolve um serviço fazendo uma requisição remota a outro serviço para recuperar dados necessários. As desvantagens desse padrão incluem latência de rede, dados e segurança, a necessidade de contratos entre os serviços, e o risco de acoplamento de serviços, onde a disponibilidade de um serviço depende da disponibilidade de outro.

4.Pergunta

Como o padrão de Replicação de Esquema de Coluna melhora o acesso a dados para microserviços?

Resposta:O padrão de Replicação de Esquema de Coluna permite que colunas de dados sejam replicadas em tabelas, tornando dados importantes acessíveis a múltiplos serviços sem requisições diretas ao serviço original. Isso melhora o desempenho de acesso, reduz a sobrecarga de comunicação e pode permitir um acesso a dados mais rápido e localizado.

5.Pergunta

Quais são as vantagens e desvantagens do uso de Cache Replicado em uma arquitetura de microserviços?



Resposta:O Cache Replicado oferece vantagens como um desempenho de acesso a dados melhorado e escalabilidade, pois os dados são armazenados em memória dentro de cada serviço. No entanto, isso também traz desvantagens, como um aumento no uso de memória dependendo do número de instâncias, possíveis desafios em manter os dados sincronizados entre os serviços, e dependências de inicialização, onde os serviços precisam estar operacionais para acessar o cache.

6.Pergunta

O que é o padrão de Domínio de Dados e por que pode ser benéfico?

Resposta:O padrão de Domínio de Dados envolve compartilhar tabelas entre serviços, o que permite acesso direto a dados sem precisar fazer chamadas remotas entre os serviços. Isso reduz dependências, melhora a consistência dos dados e simplifica o gerenciamento de fluxos de trabalho, tornando mais fácil a imposição de integridade e restrições dos dados.



7.Pergunta

Em quais cenários um arquiteto preferiria orquestração em vez de coreografia para gerenciar fluxos de trabalho?

Resposta:Um arquiteto preferiria orquestração em cenários onde os fluxos de trabalho são complexos, envolvem múltiplas condições de erro ou requerem tratamento de erros centralizado. A orquestração proporciona uma estrutura clara para gerenciar o estado e o comportamento, tornando-a adequada para lidar efetivamente com fluxos de trabalho intrincados.

8.Pergunta

Quais são as vantagens e desvantagens da coreografia na gestão de fluxos de trabalho?

Resposta:A coreografia oferece vantagens como melhor capacidade de resposta, melhor escalabilidade e menor acoplamento entre serviços. No entanto, suas desvantagens incluem potenciais dificuldades na gestão de fluxos de trabalho distribuídos, desafios na gestão de estado e a complexidade do tratamento de erros, que pode se tornar



complicada quando os fluxos de trabalho não têm um coordenador central.

Mais livros gratuitos no Bookey



Escanear para baixar



Escanear para baixar



Por que o Bookey é um aplicativo indispensável para amantes de livros



Conteúdo de 30min

Quanto mais profunda e clara for a interpretação que fornecemos, melhor será sua compreensão de cada título.



Clipes de Ideias de 3min

Impulsione seu progresso.



Questionário

Verifique se você dominou o que acabou de aprender.



E mais

Várias fontes, Caminhos em andamento, Coleções...

Teste gratuito com Bookey



Capítulo 13 | 11. Gerenciando Fluxos de Trabalho Distribuídos| Perguntas e respostas

1.Pergunta

Qual é o principal compromisso discutido ao escolher entre orquestração e coreografia em arquiteturas distribuídas?

Resposta:O principal compromisso envolve equilibrar complexidade e controle com desacoplamento e escalabilidade. A orquestração oferece controle centralizado e um tratamento de erros mais fácil, mas pode levar a gargalos e reduzir a escalabilidade. A coreografia melhora o desacoplamento e a escalabilidade, mas aumenta a complexidade na gestão de erros e na coordenação entre os serviços.

2.Pergunta

Por que os arquitetos devem evitar usar absolutos como 'sempre use coreografia' ao discutir padrões arquiteturais?

Resposta:Usar absolutos reduz a flexibilidade na tomada de



decisões arquiteturais. Cada padrão arquitetural tem seu contexto e adequação; os arquitetos devem analisar casos de uso específicos e considerar os compromissos em vez de aderir a regras rígidas.

3.Pergunta

Quais são as vantagens e desvantagens do estilo de comunicação por orquestração?

Resposta:As vantagens incluem gerenciamento centralizado de fluxo de trabalho, tratamento de erros, recuperabilidade e gerenciamento de estado administrável. As desvantagens envolvem gargalos de responsividade, potenciais pontos únicos de falha e desafios na escalabilidade devido ao aumento dos pontos de coordenação.

4.Pergunta

Em quais cenários a coreografia pode ser preferível à orquestração?

Resposta:A coreografia é preferível em cenários que exigem alta responsividade e escalabilidade, tipicamente com fluxos de trabalho mais simples ou quando condições de erro são



infrequentes, permitindo que os serviços operem de forma independente e mais eficiente.

5.Pergunta

Quais são os desafios comuns que atualizações compensatórias enfrentam em transações distribuídas?

Resposta:Atualizações compensatórias podem falhar ao serem executadas corretamente se os serviços não conseguirem reverter alterações feitas durante a transação inicial, levando a inconsistências ou confusão no estado do sistema. Elas também carecem de isolamento de transação, o que pode causar efeitos colaterais e complicar o tratamento de erros.

6.Pergunta

Como uma máquina de estados ajuda na gestão de sagas transacionais?

Resposta:Uma máquina de estados pode rastrear o estado atual de uma saga e supervisionar as transições entre os estados com base em eventos e ações, facilitando uma melhor gestão dos fluxos de trabalho e proporcionando



clareza sobre onde os erros podem precisar ser tratados.

7.Pergunta

Quais são alguns dos melhores casos de uso para o padrão Anthology Saga(aec)?

Resposta:O padrão Anthology Saga(aec) é bem adequado para cargas de trabalho de alta vazão que envolvem condições de erro simples ou infrequentes, como processos de ingestão de dados ou sistemas que utilizam uma arquitetura de Tubos e Filtros, onde flexibilidade e escalabilidade são priorizadas.

8.Pergunta

Por que é importante para um arquiteto entender e comunicar os compromissos associados a diferentes padrões de saga?

Resposta:Compreender os compromissos ajuda os arquitetos a tomar decisões informadas que estejam alinhadas com os requisitos de negócio e restrições técnicas, garantindo que a solução escolhida seja a mais adequada para o fluxo de trabalho específico e reduzindo os riscos associados ao desalinhamento do projeto.



9.Pergunta

Qual é o papel do contexto no design de fluxos de trabalho em um sistema que utiliza microserviços?

Resposta:O contexto é crucial, pois define como os serviços interagem, o significado semântico por trás das mudanças de estado e como o tratamento de erros é abordado. Um contexto gerido corretamente pode prevenir o acoplamento e a complexidade desnecessários, melhorando a manutenibilidade e a agilidade.

Capítulo 14 | 12. Sagases Transacionais| Perguntas e respostas

1.Pergunta

Qual é o padrão de comunicação da saga Horror Story e por que é considerado um anti-padrão?

Resposta:O padrão de saga Horror Story é caracterizado pela comunicação assíncrona, consistência atômica e coordenação coreografada. É considerado um anti-padrão porque combina o requisito mais rigoroso (consistência atômica) com os estilos de acoplamento mais flexíveis (assíncrono e



coreografia). Isso torna a gestão da integridade transacional excessivamente complexa, especialmente quando ocorrem erros, uma vez que cada serviço deve rastrear seu estado e a ordem de execução de forma independente, sem um orquestrador central.

2.Pergunta

Como o padrão Phone Tag Saga(sac) melhora características arquitetônicas como escalabilidade?

Resposta:O padrão Phone Tag Saga(sac) melhora a escalabilidade ao utilizar coreografia em vez de orquestração, o que reduz gargalos. Em condições normais, isso permite um maior throughput, pois o último serviço no fluxo de trabalho pode retornar um resultado sem esperar por um mediador. Isso leva a menos pontos de estrangulamento, melhorando assim o desempenho geral ao seguir um caminho feliz.

3.Pergunta

Quais desafios surgem quando atualizações compensatórias falham em um fluxo de trabalho do



padrão Epic Saga(sao)?

Resposta: Quando atualizações compensatórias falham em um fluxo de trabalho da Epic Saga(sao), isso cria confusão sobre a resposta a ser enviada de volta ao usuário final. Por exemplo, se um ticket já está marcado como completo e a atualização compensatória falha, emitir outro pedido de 'marcar como completo' pode resultar em erros adicionais. Além disso, podem surgir efeitos colaterais, onde ações realizadas por outros serviços com base no estado inicial não podem ser revertidas, contribuindo para a complexidade operacional e a inconsistência.

4.Pergunta

Quais são as vantagens de usar o padrão Fairy Tale Saga(seo)?

Resposta: O padrão Fairy Tale Saga(seo) oferece melhor responsividade, pois se baseia na consistência eventual em vez de estrita atomicidade. Isso permite que cada serviço gerencie suas próprias transações de forma mais flexível, facilitando o manuseio de indisponibilidades temporárias do



serviço. O padrão também possibilita o tratamento assíncrono de erros sem afetar a experiência do usuário.

5.Pergunta

Por que a Anthology Saga(aec) é considerada altamente desacoplada e quais são seus principais desafios?

Resposta:O padrão Anthology Saga(aec) é altamente desacoplado porque utiliza comunicação assíncrona e consistência eventual sem um orquestrador central. Isso permite melhor escalabilidade e responsividade. No entanto, os desafios incluem a gestão da coordenação e o tratamento de erros, uma vez que cada serviço deve manter estado e lidar com erros de forma autônoma, o que pode levar a uma complexidade aumentada nos fluxos de trabalho.

6.Pergunta

Qual é o papel das transações compensatórias dentro dos padrões de saga?

Resposta:As transações compensatórias são usadas para reverter alterações feitas durante a execução de uma saga quando ocorre um erro. Elas tentam restaurar o estado



anterior do sistema emitindo atualizações corretivas para os serviços envolvidos, abordando assim as inconsistências causadas por transações falhadas. No entanto, gerenciar essas compensações adiciona complexidade e pode levar a mais condições de erro.

7.Pergunta

Por que um arquiteto poderia escolher o padrão Parallel Saga(aeo) em vez do padrão Epic Saga(sao)?

Resposta:Um arquiteto pode escolher o padrão Parallel Saga(aeo) em vez do padrão Epic Saga(sao) porque ele afrouxa restrições ao utilizar comunicação assíncrona e consistência eventual, o que melhora a responsividade e a escalabilidade. A Parallel Saga pode lidar de forma eficiente com fluxos de trabalho complexos sem ser engarrafada por operações síncronas, permitindo a execução paralela de serviços.

8.Pergunta

Qual é o impacto da coordenação transacional no desempenho do sistema em ARQUITETURA DE SOFTWARE?



Resposta:A coordenação transacional em ARQUITETURA DE SOFTWARE pode afetar significativamente o desempenho do sistema. Frequentemente, ela cria gargalos, especialmente em ambientes de alta concorrência, devido à necessidade de os serviços sincronizarem e gerenciarem estados em transações distribuídas. Isso pode levar a menor responsividade, redução da escalabilidade e maior complexidade no tratamento de erros.

9.Pergunta

Como as máquinas de estado facilitam a gestão de sagas transacionais?

Resposta:As máquinas de estado facilitam a gestão de sagas transacionais definindo todos os possíveis estados e transições dentro de um fluxo de trabalho. Elas ajudam a acompanhar o estado atual e possibilitam o tratamento sistemático de transições e condições de erro. Essa abordagem estruturada auxilia na implementação de fluxos de trabalho robustos e garante que todos os caminhos potenciais, incluindo os estados de erro, sejam levados em



consideração.

10.Pergunta

Em quais cenários o padrão Time Travel Saga(sec) é mais eficaz?

Resposta:O padrão Time Travel Saga(sec) é mais eficaz em fluxos de trabalho simples onde a complexidade da orquestração é baixa e onde são exigidos alto throughput e eficiência, como em processamento de dados em massa ou cenários onde operações de 'fire and forget' são aceitáveis. Ele fornece um bom equilíbrio para fluxos de trabalho que não requerem coordenação transacional rigorosa, permitindo que cada serviço gerencie seu próprio estado e responsabilidade.

Capítulo 15 | 13. Contratos| Perguntas e respostas

1.Pergunta

Como determinamos se devemos usar contratos rígidos ou flexíveis em nossa ARQUITETURA DE SOFTWARE?

Resposta:A decisão depende da frequência de mudanças, do nível de acoplamento semântico e dos



trade-offs entre rigidez e flexibilidade. Contratos rígidos oferecem fidelidade garantida e verificação mais fácil, mas criam um acoplamento estreito, enquanto contratos flexíveis permitem maior liberdade e desacoplamento, à custa de possíveis desafios de integração.

2.Pergunta

Qual é a principal diferença entre os padrões de Data Warehouse e Data Lake para gerenciar dados analíticos?

Resposta:O Data Warehouse foca na extração e transformação de dados operacionais em um esquema estruturado para análise, enquanto o Data Lake enfatiza o armazenamento de dados em sua forma bruta e a transformação deles apenas quando necessário, permitindo análises mais flexíveis e sob demanda.

3.Pergunta

Quais problemas surgem do acoplamento de carimbo em ARQUITETURAS DISTRIBUÍDAS?

Resposta:O acoplamento de carimbo pode levar à fragilidade



quando detalhes desnecessários são incluídos nos contratos, impactando a agilidade do sistema. Também arrisca o uso excessivo de largura de banda ao passar grandes estruturas de dados que contêm mais informações do que o necessário pelo serviço receptor.

4.Pergunta

O que é um contrato dirigido pelo consumidor e por que é benéfico?

Resposta:Um contrato dirigido pelo consumidor permite que o consumidor especifique quais dados precisa do provedor, garantindo que o provedor possa atender a essas necessidades sem acoplá-los rigidamente. Essa abordagem melhora a flexibilidade e o desacoplamento, facilitando a gestão de mudanças.

5.Pergunta

Quais são os principais princípios que fundamentam a ARQUITETURA DE DATA MESH?

Resposta:A ARQUITETURA DE DATA MESH é construída sobre quatro princípios fundamentais: propriedade do



domínio sobre os dados (dados geridos pelos domínios que melhor os compreendem), tratar os dados como um produto, ter uma plataforma de dados autosserviço para as equipes de domínio e implementar governança federada computacional para garantir qualidade e conformidade consistentes dos dados.

6.Pergunta

Por que é importante garantir instantâneas diárias completas no DPQ de Suprimento Especialista?

Resposta:Instantâneas diárias completas são essenciais para manter a integridade da análise de tendências; dados incompletos podem distorcer resultados, levando a decisões equivocadas. Portanto, deve-se processar ou um conjunto de dados completo do dia ou nenhum.

7.Pergunta

Como o padrão Data Mesh aborda as limitações dos modelos tradicionais de gerenciamento de dados?

Resposta:O padrão Data Mesh supera as limitações de modelos de dados centralizados, como Data Warehouses e



Data Lakes, promovendo a propriedade descentralizada de dados alinhada aos domínios de negócios, fomentando a propriedade direta e permitindo análises mais relevantes e oportunas.

8.Pergunta

No contexto de contratos, o que significa 'fidelidade contratual garantida'?

Resposta:Fidelidade contratual garantida refere-se à garantia de que os dados trocados entre os sistemas aderem estritamente aos esquemas definidos, incluindo nomes e tipos, prevenindo ambiguidades e erros de integração.

9.Pergunta

Quais devem ser as consequências se houver muitos dias isentos na análise de tendências?

Resposta:Se muitos dias se tornarem isentos da análise de dados devido a informações incompletas, a precisão geral da análise de tendências diminuirá, impactando negativamente a tomada de decisões relacionadas ao planejamento de suprimentos e alocação de recursos.



10.Pergunta

Qual abordagem técnica é sugerida para lidar com comunicação de dados assíncrona na ARQUITETURA DE DATA MESH?

Resposta: Recomenda-se implementar padrões de comunicação que apresentem consistência eventual e assíncronia para desacoplar operações, garantindo a qualidade dos dados em serviços e análises.



Ad



Escanear para baixar



App Store
Escolha dos Editores



22k avaliações de 5 estrelas

Feedback Positivo

Afonso Silva

...cada resumo de livro não só
..., mas também tornam o
...divertido e envolvente. O
...tizou a leitura para mim.

Fantástico!



Estou maravilhado com a variedade de livros e idiomas
que o Bookey suporta. Não é apenas um aplicativo, é
um portal para o conhecimento global. Além disso,
ganhar pontos para caridade é um grande bônus!

Brígida Santos

F



O
só
o
O

na Oliveira

...correr as
...ém me dá
...omprar a
...ar!

Adoro!



Usar o Bookey ajudou-me a cultivar um hábito de
leitura sem sobrecarregar minha agenda. O design do
aplicativo e suas funcionalidades são amigáveis,
tornando o crescimento intelectual acessível a todos.

Duarte Costa

Economiza tempo!



O Bookey é o meu apli
crescimento intelectual
perspicazes e lindame
um mundo de conheci

Aplicativo incrível!



Eu amo audiolivros, mas nem sempre tenho tempo para
ouvir o livro inteiro! O Bookey permite-me obter um resumo
dos destaques do livro que me interessa!!! Que ótimo
conceito!!! Altamente recomendado!

Estevão Pereira

Aplicativo lindo



Este aplicativo é um salva-vidas para
de livros com agendas lotadas. Os re
precisos, e os mapas mentais ajudar
o que aprendi. Altamente recomend

Teste gratuito com Bookey



Capítulo 16 | 14. Gerenciando Dados Analíticos| Perguntas e respostas

1.Pergunta

Qual é o problema que surge da divisão entre dados operacionais e analíticos em arquiteturas modernas?

Resposta:As arquiteturas modernas enfrentam dificuldades devido às diferenças nos formatos e nas exigências de esquema para dados operacionais e analíticos, tornando desafiador o uso eficaz de ambos. As exigências analíticas, como agregações ou consultas complexas, muitas vezes não podem ser facilmente atendidas por sistemas operacionais transacionais.

2.Pergunta

Quais são as principais características do padrão Data Warehouse?

Resposta:O padrão Data Warehouse inclui a extração de dados de várias fontes para um repositório central, transformando-os em um único esquema, geralmente utilizando o esquema em estrela, carregando-os no armazém



para análise e permitindo que analistas de dados realizem relatórios baseados nesses dados.

3.Pergunta

Quais são algumas falhas do padrão Data Warehouse?

Resposta:As falhas incluem a rigidez da integração, a extremada partição do conhecimento de domínio, complexidade adicional, funcionalidade limitada para seu propósito pretendido e gargalos de sincronização que podem prejudicar a eficiência operacional.

4.Pergunta

Como o padrão Data Lake difere do padrão Data Warehouse?

Resposta:O padrão Data Lake adota uma abordagem de 'carregar e transformar' em vez de 'transformar e carregar', permitindo que dados brutos sejam armazenados sem transformação extensa inicial. Isso proporciona mais flexibilidade para os cientistas de dados acessarem dados conforme necessário, principalmente para casos de uso de aprendizado de máquina.



5.Pergunta

Quais são os quatro princípios em que o Data Mesh é fundado?

Resposta: 1. Propriedade de domínio dos dados, assegurando que os domínios compartilhem a responsabilidade por seus dados. 2. Dados como um produto, incentivando os domínios a fornecerem dados aos consumidores de forma eficaz. 3. Capacidades de plataforma de dados self-serve para capacitar equipes a construir e gerenciar seus produtos de dados. 4. Governança federada computacional para atender aos requisitos de governança em toda a organização de maneira consistente em todos os domínios.

6.Pergunta

O que representa o Quantum de Produto de Dados (DPQ) em uma arquitetura Data Mesh?

Resposta: O Quantum de Produto de Dados (DPQ) atua como um componente altamente acoplado, mas operacionalmente independente, associado a um serviço, responsável por gerenciar dados analíticos e de relatórios dentro de um



domínio, garantindo que as necessidades analíticas sejam atendidas sem comprometer a eficiência operacional.

7.Pergunta

Quais compensações os arquitetos devem considerar ao implementar um Data Mesh?

Resposta:Os arquitetos devem ponderar os benefícios da propriedade e acesso descentralizados dos dados em comparação com os desafios, como garantir a coordenação entre os domínios, manter a qualidade dos dados e gerenciar a conformidade com as políticas de governança.

8.Pergunta

Por que os arquitetos precisam modelar cenários de domínio relevantes durante a análise de compensações?

Resposta:Modelar cenários de domínio relevantes permite que os arquitetos compreendam as verdadeiras implicações de suas escolhas de design, focando em fatores específicos que afetam o desempenho, a extensibilidade e a manutenibilidade, em vez de se basear apenas em soluções genéricas.



9.Pergunta

O que significa o termo 'MECE' e por que é importante na análise de compensações?

Resposta:MECE significa 'mutuamente exclusivo, coletivamente exaustivo'. É importante na análise de compensações porque garante que os arquitetos comparem apenas conceitos semelhantes, cobrindo todas as opções possíveis sem sobreposições, levando a uma tomada de decisão mais clara.

10.Pergunta

Como os arquitetos podem evitar o erro de sobrecarregar as partes interessadas com detalhes técnicos?

Resposta:Os arquitetos devem se concentrar em apresentar as principais compensações e os resultados-chave relevantes para as partes interessadas, simplificando informações complexas em percepções gerenciáveis que destacam as implicações essenciais de suas decisões.

Capítulo 17 | 15. Crie Sua Própria Análise de Compromissos| Perguntas e respostas

1.Pergunta

Mais livros gratuitos no Bookey



Escanear para baixar

Qual é o principal objetivo de realizar uma análise de trade-off na ARQUITETURA DE SOFTWARE?

Resposta: O principal objetivo é analisar o impacto das decisões dentro da arquitetura, identificar dimensões entrelaçadas e avaliar como as mudanças podem afetar sistemas interdependentes. Isso permite que os arquitetos tomem decisões informadas e estratégicas que alinhem soluções técnicas com metas empresariais.

2.Pergunta

Como o foco em motores de negócios altera a abordagem para problemas técnicos?

Resposta: Ao priorizar motores de negócios, as equipes de TI começam a ver as implicações mais amplas das soluções, alinhando as tarefas técnicas com as necessidades empresariais, o que leva a resultados mais relevantes e eficazes.

3.Pergunta

Qual é o papel da colaboração na recuperação de uma arquitetura problemática?

Mais livros gratuitos no Bookey



Escanear para baixar

Resposta: A colaboração entre diferentes equipes, como as equipes de banco de dados e desenvolvimento de aplicações, promove o compartilhamento de conhecimento e a resolução conjunta de problemas, o que é essencial para migrar e melhorar a aplicação de forma eficaz.

4. Pergunta

O que são tabelas de trade-off e como elas contribuem para decisões de ARQUITETURA DE SOFTWARE informadas?

Resposta: As tabelas de trade-off representam visualmente os vários trade-offs associados a diferentes decisões arquiteturais, ajudando os arquitetos a pesar opções em relação a preocupações fundamentais como acoplamento, complexidade, capacidade de resposta e escalabilidade.

5. Pergunta

Por que é importante para os arquitetos modelar cenários prováveis antes de tomar decisões técnicas?

Resposta: Modelar cenários prováveis ajuda os arquitetos a explorar os potenciais impactos de diferentes escolhas arquiteturais, permitindo que antecipem desafios e tomem



decisões informadas com base em análises quantitativas.

6.Pergunta

Por que os arquitetos devem ter cautela ao evangelizar ferramentas ou tecnologias específicas?

Resposta:Os arquitetos devem ter cautela porque o evangelismo pode levar a percepções tendenciosas que favorecem certas ferramentas, enquanto minimizam suas desvantagens. É essencial garantir avaliações equilibradas, em vez de promover soluções que podem não se adequar a todos os cenários.

7.Pergunta

O que significa a estrutura MECE e por que é útil na tomada de decisões arquitetônicas?

Resposta:MECE significa 'Mutuamente Exclusivo, Coletivamente Exaustivo'. Essa estrutura ajuda os arquitetos a garantir que comparem componentes que não se sobrepõem e cobram todas as possibilidades, levando a decisões arquitetônicas abrangentes e fundamentadas.

8.Pergunta

Como o design iterativo beneficia a ARQUITETURA DE

Mais livros gratuitos no Bookey



Escanear para baixar

SOFTWARE?

Resposta: O design iterativo permite que os arquitetos refinem continuamente as soluções, testem diferentes cenários e melhorem gradualmente sua arquitetura com base em evidências empíricas em vez de suposições, promovendo adaptabilidade e precisão na tomada de decisões.

9.Pergunta

Qual é o aprendizado enfatizado em relação ao contexto em que as decisões arquitetônicas são tomadas?

Resposta: O contexto deve ser cuidadosamente considerado, pois pode influenciar drasticamente os trade-offs e o sucesso final de uma solução. Os arquitetos devem evitar tomar decisões isoladamente, integrando fatores relevantes para guiar sua análise.

10.Pergunta

Qual é o último conselho dado sobre a abordagem das decisões arquitetônicas em equipe?

Resposta: Os arquitetos devem evitar serem compelidos a defender tecnologias ou processos específicos sem uma



análise minuciosa. Em vez disso, devem centrar as discussões em torno dos trade-offs, garantindo que perspectivas diversas sejam consideradas na tomada de decisões.

Capítulo 18 | A. Referências de Conceitos e Termos| Perguntas e respostas

1.Pergunta

O que é complexidade ciclomática e por que é importante na ARQUITETURA DE SOFTWARE?

Resposta:A complexidade ciclomática, discutida no Capítulo 6 do livro anterior, mede o número de caminhos linearmente independentes através do código fonte de um programa. É importante porque ajuda a avaliar a complexidade de um programa, facilitando a identificação de áreas que podem exigir refatoração ou testes adicionais.

2.Pergunta

Como o acoplamento de componentes afeta a ARQUITETURA DE SOFTWARE?

Resposta:O acoplamento de componentes, delineado no



Capítulo 7, refere-se ao grau de interdependência entre componentes. Um baixo acoplamento é preferido, pois permite modificações mais fáceis e aprimora a manutenibilidade do sistema, enquanto um alto acoplamento pode levar a dificuldades na gestão de mudanças e na compreensão da ARQUITETURA DE SOFTWARE como um todo.

3.Pergunta

Você pode explicar a diferença entre particionamento técnico e de domínio?

Resposta:O particionamento técnico versus de domínio é detalhado no Capítulo 8. O particionamento técnico divide componentes com base em preocupações técnicas (como pilha de tecnologia ou infraestrutura), enquanto o particionamento de domínio divide componentes com base em domínios de negócios ou funcionalidades. Escolher a abordagem certa pode impactar significativamente a escalabilidade e manutenibilidade do sistema.

4.Pergunta



Qual é o papel da ARQUITETURA DE SOFTWARE em design de software?

Resposta:A ARQUITETURA DE SOFTWARE, discutida no Capítulo 10, organiza uma aplicação de software em camadas, onde cada camada tem suas responsabilidades específicas. Essa estrutura promove separação de preocupações, tornando mais fácil gerenciar a complexidade e permitindo que equipes trabalhem em diferentes camadas sem afetar as outras.

5.Pergunta

Como os microserviços melhoram o desenvolvimento de software?

Resposta:A ARQUITETURA DE MICROSERVIÇOS, explicada no Capítulo 12, divide aplicações em serviços menores, independentes e implantáveis. Isso melhora o desenvolvimento de software ao permitir que equipes desenvolvam, testem e implantem serviços de forma independente, melhorando a escalabilidade, flexibilidade e resiliência.



6.Pergunta

O que é um Registro de Decisões Arquitetônicas (ADR) e sua importância?

Resposta:Um Registro de Decisões Arquitetônicas (ADR) é um documento que captura decisões arquitetônicas significativas, juntamente com seu contexto e consequências. Eles são vitais para manter um histórico claro de escolhas arquitetônicas, facilitando para as equipes entender as decisões tomadas e a lógica por trás delas.

7.Pergunta

Você pode dar um exemplo de uma decisão arquitetônica específica que pode ser registrada em um ADR?

Resposta:Um exemplo da lista é 'ADR: Migrar a Aplicação do Squad de Sysops para uma ARQUITETURA DISTRIBUÍDA.' Esta decisão descreve o contexto para a migração, as implicações para a estrutura da aplicação e benefícios esperados, como melhor escalabilidade e confiabilidade.

8.Pergunta

Como o uso de um banco de dados de documentos para

Mais livros gratuitos no Bookey



Escanear para baixar

pesquisas com clientes beneficia a ARQUITETURA DE SOFTWARE?

Resposta: Usar um banco de dados de documentos para pesquisas com clientes, conforme mencionado em um ADR, permite um armazenamento e recuperação de dados flexível, capaz de lidar com estruturas de pesquisa variadas ao longo do tempo. Essa flexibilidade suporta mudanças rápidas nas necessidades de negócios sem extensas modificações na ARQUITETURA DE SOFTWARE backend.

9.Pergunta

Quais são as considerações para usar cache replicado em memória?

Resposta: A escolha de usar cache replicado em memória, conforme referenciado no ADR, foca em otimizar os tempos de resposta e reduzir a latência para dados acessados frequentemente, como perfis de especialistas. Isso pode melhorar significativamente o desempenho da aplicação, especialmente em cenários de alta carga.

10.Pergunta



Por que a orquestração é utilizada para fluxos de trabalho principais de tickets?

Resposta: Usar orquestração para fluxos de trabalho principais de tickets, conforme mencionado no ADR, permite uma melhor gestão e monitoramento dos processos de negócios. Ela proporciona clareza sobre o fluxo do ciclo de vida do ticket, garantindo que todas as etapas do fluxo de trabalho sejam executadas de maneira correta e eficiente.

Mais livros gratuitos no Bookey



Escanear para baixar



Ler, Compartilhar, Empoderar

Conclua Seu Desafio de Leitura, Doe Livros para Crianças Africanas.

O Conceito



Esta atividade de doação de livros está sendo realizada em conjunto com a Books For Africa. Lançamos este projeto porque compartilhamos a mesma crença que a BFA: Para muitas crianças na África, o presente de livros é verdadeiramente um presente de esperança.

A Regra



Ganhe 100 pontos



Resgate um livro



Doe para a África

Seu aprendizado não traz apenas conhecimento, mas também permite que você ganhe pontos para causas beneficentes! Para cada 100 pontos ganhos, um livro será doado para a África.

Teste gratuito com Bookee



Capítulo 19 | B. Referências de Registro de Decisões de Arquitetura| Perguntas e respostas

1.Pergunta

Qual é o propósito dos Registros de Decisões Arquitetônicas (ADRs) dentro da ARQUITETURA DE SOFTWARE?

Resposta:Os Registros de Decisões Arquitetônicas servem como um log histórico que captura a razão por trás das principais decisões arquitetônicas tomadas durante o processo de desenvolvimento. Ao documentar essas decisões, as equipes podem manter clareza sobre o motivo de escolhas particulares terem sido feitas, facilitando melhor compreensão e revisões futuras, se necessário.

2.Pergunta

Você pode explicar a importância da análise de trade-offs no design arquitetônico?

Resposta:A análise de trade-offs é crucial na ARQUITETURA DE SOFTWARE, pois permite que as equipes comparem várias opções arquitetônicas em relação a



preocupações específicas, como desempenho, manutenibilidade e escalabilidade. Ao avaliar sistematicamente os benefícios e desvantagens de cada opção, os arquitetos podem tomar decisões informadas que alinhem-se aos objetivos e restrições do projeto.

3.Pergunta

Como o uso de um banco de dados de documentos para pesquisas de clientes ilustra a tomada de decisão arquitetônica?

Resposta:A escolha de usar um banco de dados de documentos para pesquisas de clientes destaca a necessidade de estruturas de dados flexíveis que possam acomodar formatos variados de pesquisa. Essa decisão reflete uma compreensão dos requisitos do projeto para agilidade no manuseio de dados não estruturados, demonstrando consideração cuidadosa tanto pelas necessidades de armazenamento de dados quanto pela experiência do usuário.

4.Pergunta

Quais são os desafios de migrar para uma arquitetura distribuída, conforme mencionado nos exemplos de ADR?



Resposta:Migrar para uma arquitetura distribuída apresenta vários desafios, incluindo aumento da complexidade na gestão de dados, potencial de latência na rede e a necessidade de uma orquestração eficaz dos serviços. Esses desafios exigem planejamento e consideração cuidadosos, equilibrando os benefícios de escalabilidade e tolerância a falhas com o overhead operacional.

5.Pergunta

Por que é importante ter trade-offs documentados e referenciados no processo de desenvolvimento?

Resposta:Documentar trade-offs é essencial porque cria uma base para entender as implicações das decisões arquitetônicas. Quando esses trade-offs são claramente referenciados, permite que as equipes justifiquem suas escolhas, avaliem sua validade ao longo do tempo e ofereçam um recurso de aprendizado para futuros esforços de desenvolvimento.

6.Pergunta

Qual é o papel do design de contrato solto em uma arquitetura de microserviços, com base nos ADRs?



Resposta: O design de contrato solto na arquitetura de microserviços facilita maior flexibilidade e independência entre os serviços. Essa abordagem permite uma evolução mais fácil dos serviços individuais, reduzindo o impacto das mudanças no sistema geral e aumentando a agilidade no processo de desenvolvimento.

7. Pergunta

Como a escolha de soluções de armazenamento de dados pode afetar o desempenho geral da aplicação?

Resposta: A escolha de soluções de armazenamento de dados impacta diretamente o desempenho da aplicação ao influenciar fatores como velocidade de recuperação, escalabilidade sob carga e facilidade de integração com outros serviços. Por exemplo, usar bancos de dados em memória pode melhorar significativamente a velocidade para dados acessados frequentemente em comparação com soluções tradicionais baseadas em disco.

8. Pergunta

Que insights podem ser extraídos dos trade-offs relacionados à gestão de estado versus transações



atômicas?

Resposta: Os insights extraídos da comparação entre técnicas de gestão de estado e transações atômicas destacam a necessidade de equilibrar a consistência dos dados com o desempenho. Embora transações atômicas garantam a integridade dos dados, elas podem introduzir latência; por outro lado, abordagens alternativas de gestão de estado podem oferecer melhor capacidade de resposta, mas exigem um tratamento mais complexo para manter a consistência.

9. Pergunta

Qual a importância da documentação dos padrões de acoplamento operacional para o futuro desenvolvimento de software?

Resposta: Documentar os padrões de acoplamento operacional é vital para o futuro desenvolvimento de software, pois permite que as equipes entendam as interações e dependências entre diferentes serviços. Essa visão é crucial para diagnosticar problemas, otimizar o desempenho e planejar a escalabilidade em arquiteturas em evolução.



10.Pergunta

Como as decisões arquitetônicas em torno de padrões de serviço mesh influenciam a colaboração da equipe?

Resposta:As decisões arquitetônicas sobre padrões de serviço mesh podem melhorar a colaboração da equipe ao padronizar protocolos de comunicação e simplificar interações entre serviços. Essa padronização pode reduzir conflitos entre as equipes, aumentar a autonomia e agilizar os processos de desenvolvimento e implantação em microserviços.

Capítulo 20 | C. Referências de Trade-Off| Perguntas e respostas

1.Pergunta

Qual é o propósito de incluir tabelas e figuras de trade-off nas discussões sobre ARQUITETURA DE SOFTWARE?

Resposta:O principal propósito dessas tabelas e figuras de trade-off é resumir preocupações arquitetônicas chave e facilitar a análise de trade-off, o que permite que os arquitetos de software tomem decisões informadas sobre o design e a implementação de um sistema.



2.Pergunta

Como os diversos tipos de banco de dados, como relacionais e NoSQL, diferem nas características de adoção?

Resposta:Cada tipo de banco de dados possui características de adoção únicas que o tornam adequado para casos de uso específicos. Por exemplo, bancos de dados relacionais costumam se destacar na conformidade com ACID, tornando-os ideais para aplicações com alta carga de transações, enquanto bancos NoSQL como armazenamento de chave-valor e documentos oferecem flexibilidade e escalabilidade para modelos de dados menos estruturados.

3.Pergunta

Quais são os trade-offs envolvidos no uso da 'replicação de código' como técnica de reuso?

Resposta:Os trade-offs da replicação de código incluem aumento da redundância, potencial para problemas de versionamento e dificuldades na manutenção, em contrapartida ao benefício de uma sobrecarga de dependência reduzida e melhor desempenho devido a cópias localizadas.



4.Pergunta

Qual é a importância do trade-off entre orquestração e coreografia na comunicação de serviços?

Resposta:O trade-off entre orquestração e coreografia determina como os serviços distribuídos interagem entre si, influenciando a complexidade do sistema, escalabilidade e recuperação de falhas. A orquestração centraliza o controle, facilitando a gestão, enquanto a coreografia oferece uma abordagem mais descentralizada que pode aumentar a resiliência e a flexibilidade.

5.Pergunta

Por que uma equipe poderia preferir acoplamento dinâmico em vez de acoplamento estático nas interações de componentes?

Resposta:O acoplamento dinâmico permite que sistemas conectem componentes de forma adaptativa durante a execução, o que pode aumentar a flexibilidade e reduzir o impacto de mudanças na ARQUITETURA DE SOFTWARE como um todo. Em contraste, o acoplamento estático pode levar a arquiteturas mais rígidas.



6.Pergunta

Como os tipos de banco de dados impactam o desempenho e a escalabilidade do sistema?

Resposta:Diferentes tipos de banco de dados oferecem características variáveis de desempenho e escalabilidade. Por exemplo, bancos de dados relacionais podem ter dificuldade em escalar horizontalmente sob altas cargas devido a seus requisitos de transação ACID, enquanto bancos NoSQL são projetados para escalabilidade horizontal e podem lidar com grandes volumes de dados não estruturados.

7.Pergunta

O que as funções de fitness representam na tomada de decisões sobre ARQUITETURA DE SOFTWARE?

Resposta:As funções de fitness são métricas que ajudam a avaliar a qualidade da ARQUITETURA e seu alinhamento com os objetivos de negócios, assegurando que a ARQUITETURA não apenas atenda a padrões técnicos, mas também suporte os objetivos desejados de desempenho, manutenibilidade e escalabilidade.



8.Pergunta

Em quais cenários os contratos estritos seriam preferidos em relação aos contratos frouxos em microserviços?

Resposta: Contratos estritos são preferidos em casos onde a confiabilidade, desempenho e segurança do sistema são críticas, pois minimizam ambiguidade e garantem um controle mais rígido sobre as interações entre microserviços, reduzindo o risco de mudanças quebradoras.

9.Pergunta

Qual é o papel da propriedade dos dados no design arquitetônico?

Resposta: A propriedade dos dados é crucial para definir responsabilidades claras na gestão de dados entre os serviços, facilitando melhor modularidade e reduzindo o acoplamento, além de agilizar a governança e conformidade dos dados dentro da ARQUITETURA.

10.Pergunta

Por que o processo de análise iterativa é importante na análise de trade-off?

Resposta: A análise iterativa incentiva o refinamento contínuo



das decisões arquitetônicas, permitindo que as equipes se adaptem a novas informações, requisitos em mudança ou desafios imprevistos, melhorando assim a robustez e adaptabilidade geral do design arquitetônico.

Mais livros gratuitos no Bookey



Escanear para baixar



As melhores ideias do mundo desbloqueiam seu potencial

Essai gratuit avec Bookey



Escanear para baixar



ARQUITETURA DE SOFTWARE Quiz e teste

Ver a resposta correta no site do Bookey

Capítulo 1 | 1. O Que Acontece Quando Não Há “Melhores Práticas”?| Quiz e teste

- 1.Os tecnólogos são vistos cada vez mais como fontes confiáveis para as melhores práticas em ARQUITETURA DE SOFTWARE.
- 2.Os arquitetos frequentemente enfrentam desafios únicos que foram documentados anteriormente e têm soluções claras.
- 3.Os Registros de Decisão Arquitetônica (ADRs) são úteis para documentar decisões arquitetônicas e suas consequências.

Capítulo 2 | I. Desmontando as Coisas| Quiz e teste

- 1.O acoplamento estático refere-se a como partes da arquitetura, como componentes, estão conectadas antes da execução e podem ser examinadas durante a execução.

Mais livros gratuitos no Bookey



Escanear para baixar

2. Alta coesão funcional indica que os elementos da arquitetura trabalham juntos de forma eficaz e geralmente representam um domínio ou fluxo de trabalho.
3. O acoplamento dinâmico descreve como partes da arquitetura estão conectadas antes da execução e não envolve interações em tempo de execução.

Capítulo 3 | 2. Discernindo o Acoplamento na ARQUITETURA DE SOFTWARE| Quiz e teste

1. A modularidade arquitetônica melhora o desempenho, a manutenibilidade e a escalabilidade do sistema.
2. As arquiteturas monolíticas são mais adaptáveis a mudanças rápidas nas demandas de negócios do que as arquiteturas modulares.
3. A melhoria da testabilidade é um dos benefícios das arquiteturas modulares, permitindo a testagem independente de serviços.



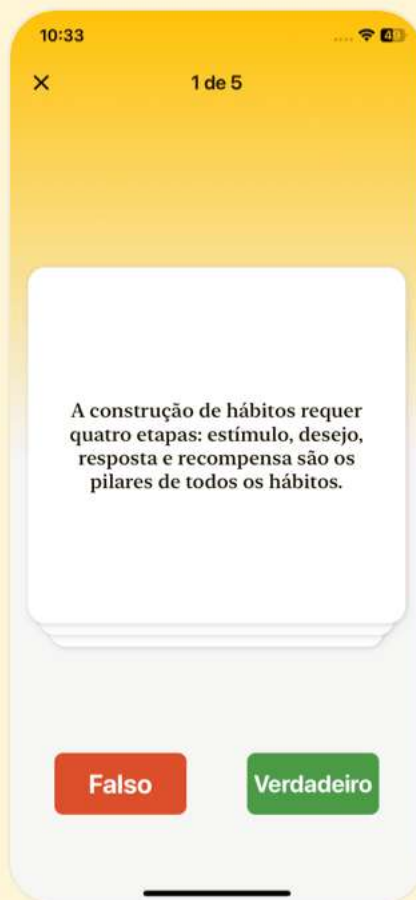
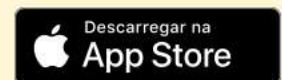


Baixe o app Bookey para desfrutar

Mais de 1000 resumos de livros com quizzes

Teste grátis disponível!

Escanear para baixar



Capítulo 4 | 3. Modularidade Arquitetônica| Quiz e teste

1. Addison e Austen decidiram começar sua decomposição arquitetônica abordando a função de relatórios primeiro.
2. Logan aconselhou a utilização de uma abordagem não estruturada para migração conhecida como 'forking tático'.
3. A decomposição baseada em componentes é mais adequada para bases de código bem estruturadas com componentes identificáveis.

Capítulo 5 | 4. Decomposição Arquitetônica| Quiz e teste

1. O padrão 'Identificar e Dimensionar Componentes' foca em identificar componentes para garantir uma arquitetura modular.
2. O padrão 'Coletar Componentes de Domínio Comuns' visa aumentar a redundância ao separar serviços duplicados entre componentes.
3. O padrão 'Criar Serviços de Domínio' diz respeito a mover grupos de domínio para aplicações monolíticas para manter



a estrutura existente.

Capítulo 6 | 5. Padrões de Decomposição Baseados em Componentes| Quiz e teste

1.O padrão 'Identificar e Dimensionar

Componentes' garante que os componentes sejam dimensionados adequadamente para evitar alto acoplamento e permitir a separação de serviços.

2.O padrão 'Reunir Componentes Comuns de Domínio' visa criar duplicatas de funcionalidades para melhorar a criação de serviços.

3.O padrão 'Criar Serviços de Domínio' marca a transição para uma arquitetura baseada em serviços ao extrair componentes em serviços implantados separadamente.



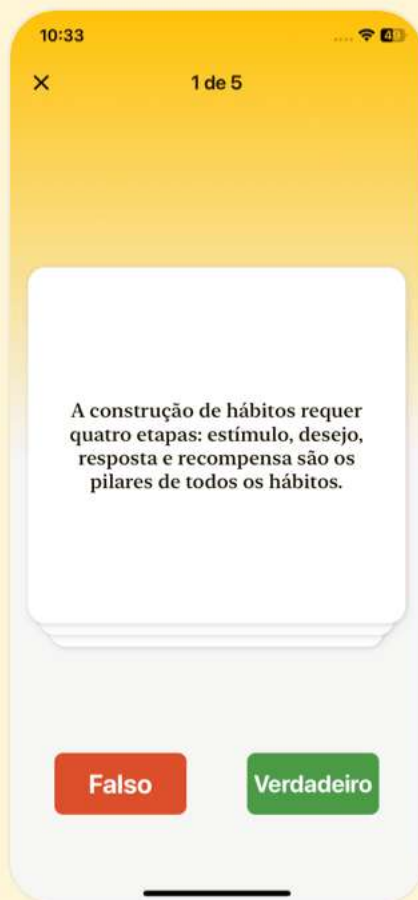


Baixe o app Bookey para desfrutar

Mais de 1000 resumos de livros com quizzes

Teste grátis disponível!

Escanear para baixar



Capítulo 7 | 6. Desmembrando Dados Operacionais| Quiz e teste

- 1.O capítulo argumenta que gerenciar mudanças no esquema do banco de dados é crucial, já que modificações podem impactar múltiplos serviços que dependem do banco de dados.
- 2.Um banco de dados compartilhado é mais resistente a falhas em comparação a vários bancos de dados independentes.
- 3.O processo de cinco etapas para decompor um banco de dados monolítico inclui a criação de domínios de dados e a movimentação de esquemas para servidores de banco de dados separados.

Capítulo 8 | 7. Granularidade de Serviço| Quiz e teste

- 1.A granularidade do serviço refere-se ao tamanho e à abrangência dos serviços individuais na ARQUITETURA DE SOFTWARE e é importante ao fazer a transição para microserviços.
- 2.De acordo com o capítulo, todas as partes de um serviço



devem sempre ser mantidas juntas para melhor desempenho e eficiência.

3.Os desintegradores de granularidade incluem fatores como volatilidade de código e tolerância a falhas, enquanto os integradores estão relacionados a transações de banco de dados e código compartilhado.

Capítulo 9 | II. Reunindo as peças| Quiz e teste

- 1.Reutilizar código em sistemas distribuídos deve ser feito com cautela, pois 'reutilização é abuso'.
- 2.A replicação de código em arquiteturas distribuídas simplifica as mudanças de código e garante consistência entre os serviços.
- 3.O padrão Sidecar permite que aspectos operacionais dos serviços sejam acoplados à lógica de domínio para melhor flexibilidade.



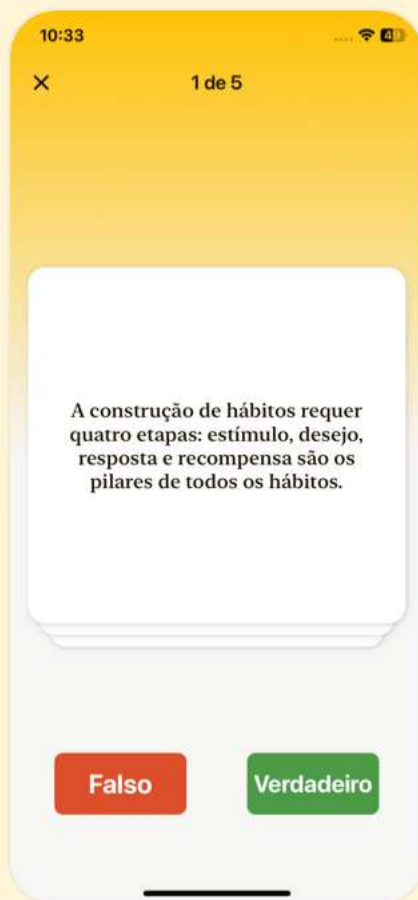


Baixe o app Bookey para desfrutar

Mais de 1000 resumos de livros com quizzes

Teste grátis disponível!

Escanear para baixar



Capítulo 10 | 8. Padrões de Reuso| Quiz e teste

1. A replicação de código é comumente usada em microserviços iniciais como uma arquitetura 'sem compartilhamento'.
2. Uma biblioteca compartilhada simplifica a gestão de dependências, mas complica o controle de mudanças.
3. O padrão Sidecar oferece uma solução para o acoplamento operacional ao separar comportamentos operacionais da lógica de domínio.

Capítulo 11 | 9. Propriedade de Dados e Transações Distribuídas| Quiz e teste

1. A posse de dados deve ser claramente atribuída aos serviços para mitigar problemas relacionados ao compartilhamento de dados em sistemas distribuídos.
2. O modelo BASE significa Disponibilidade Básica, Estado Firme e Consistência Imediata, que é usado em transações distribuídas.
3. O Padrão Baseado em Requisições Orquestradas prioriza a



responsividade dos dados em vez da consistência na gestão de transações distribuídas.

Capítulo 12 | 10. Acesso a Dados Distribuídos| Quiz e teste

- 1.O Padrão de Comunicação entre Serviços é a maneira mais comum para os serviços acessarem dados pertencentes a outros serviços em uma arquitetura distribuída.
- 2.O Padrão de Replicação de Esquema de Coluna resulta em uma melhoria no desempenho de leitura sem problemas de consistência de dados.
- 3.O Padrão de Domínio de Dados oferece excelente desempenho, mas pode introduzir preocupações de segurança devido ao acesso compartilhado ao banco de dados entre vários serviços.



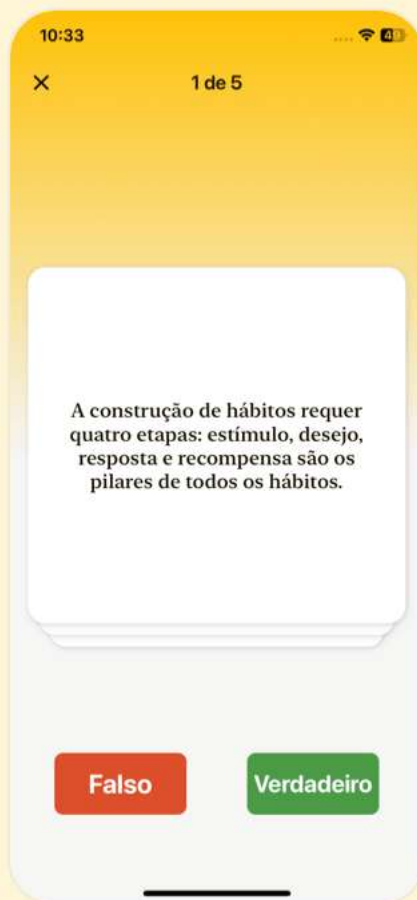


Baixe o app Bookey para desfrutar

Mais de 1000 resumos de livros com quizzes

Teste grátis disponível!

Escanear para baixar



Capítulo 13 | 11. Gerenciando Fluxos de Trabalho Distribuídos| Quiz e teste

- 1.O uso de coreografia na arquitetura elimina a necessidade de análise de compromisso.
- 2.A orquestração simplifica a lógica complexa e ajuda na gestão de erros e estado em sistemas distribuídos.
- 3.A consistência eventual garante melhor desempenho do que a atomicidade em todas as situações em sistemas distribuídos.

Capítulo 14 | 12. Sagases Transacionais| Quiz e teste

- 1.O padrão da saga Horror Story é considerado um design adequado para fluxos de trabalho transacionais na ARQUITETURA DE SOFTWARE.
- 2.As sagas são projetadas para ajudar a limitar os escopos de bloqueio de banco de dados em arquiteturas distribuídas.
- 3.O padrão da Saga Antologia é menos eficaz para fluxos de trabalho complexos devido ao seu alto nível de desacoplamento e falta de orquestração.



Capítulo 15 | 13. Contratos| Quiz e teste

- 1.Os contratos em ARQUITETURA DE SOFTWARE vêm em três formas: estritos, flexíveis e soltos.
- 2.Contratos estritos podem levar a um acoplamento forte entre serviços e trazer desafios na gestão de versões.
- 3.A arquitetura Data Mesh promove a propriedade centralizada dos dados para melhorar a gestão e análise de dados.



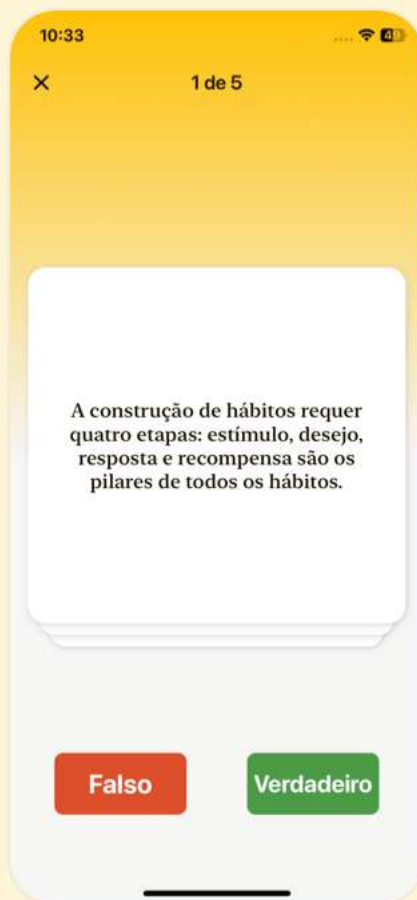


Baixe o app Bookey para desfrutar

Mais de 1000 resumos de livros com quizzes

Teste grátis disponível!

Escanear para baixar



Capítulo 16 | 14. Gerenciando Dados Analíticos| Quiz e teste

1. Um data warehouse separa dados operacionais e analíticos, mas apresenta desafios mínimos associados à sua complexidade.
2. Em uma arquitetura de data mesh, os dados são gerenciados pelos domínios mais familiarizados com eles, promovendo o acesso entre pares.
3. As arquiteturas de Data Lake requerem extensas transformações iniciais dos dados antes do armazenamento.

Capítulo 17 | 15. Crie Sua Própria Análise de Compromissos| Quiz e teste

1. Compreender os fatores de negócios deve ser priorizado em relação a questões estritamente técnicas na ARQUITETURA DE SOFTWARE.
2. A colaboração entre as equipes de TI e de aplicação não é necessária para resolver problemas nas migrações de aplicativos.
3. Cenários contínuos de 'e se' são desencorajados na análise de trade-offs porque complicam as decisões arquitetônicas.



Capítulo 18 | A. Referências de Conceitos e Termos| Quiz e teste

- 1.A complexidade ciclomática é discutida no Capítulo 6 do livro.
- 2.A arquitetura em camadas é mencionada no Capítulo 12 de 'ARQUITETURA DE SOFTWARE'.
- 3.A arquitetura de microservices é detalhada no Capítulo 13 do livro.



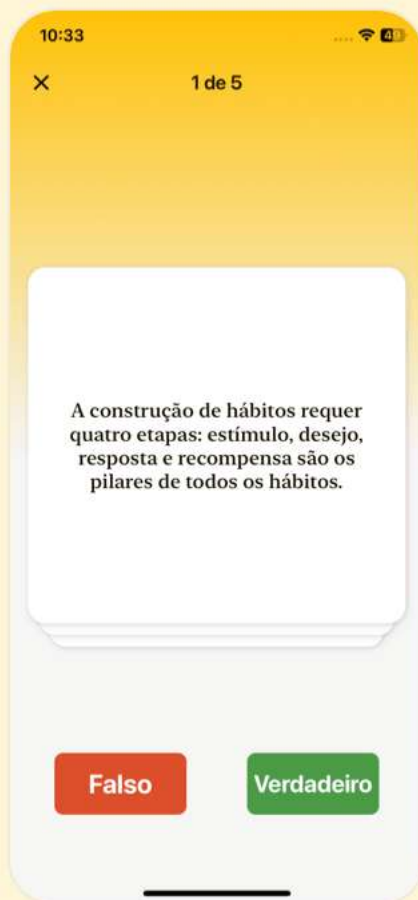


Baixe o app Bookey para desfrutar

Mais de 1000 resumos de livros com quizzes

Teste grátis disponível!

Escanear para baixar



Capítulo 19 | B. Referências de Registro de Decisões de Arquitetura| Quiz e teste

- 1.O Registro de Decisão de Arquitetura (ADR) é definido como uma frase nominal curta que contém a decisão de arquitetura.
- 2.Os Registros de Decisão de Arquitetura (ADRs) são relevantes apenas para o desenvolvimento de novas aplicações e não se aplicam a sistemas existentes.
- 3.O livro discute apenas bancos de dados relacionais como um tipo de banco de dados nas trocas arquitetônicas.

Capítulo 20 | C. Referências de Trade-Off| Quiz e teste

- 1.O apêndice C contém referências de trade-offs para vários tipos de bancos de dados, incluindo bancos de dados relacionais e de grafos.
- 2.A Tabela 12-11 discute os trade-offs associados à gestão de estado versus transações distribuídas atômicas com atualizações compensatórias.
- 3.O livro fornece apenas uma tabela dedicada a trade-offs para orquestração e coreografia dentro dos fluxos de



trabalho de bilhetes.

Mais livros gratuitos no Bookey



Escanear para baixar



Baixe o app Bookey para desfrutar

Mais de 1000 resumos de livros com quizzes

Teste grátis disponível!

Escanear para baixar

