

MICROSERVIÇOS

Vinicius Soares e Luis Fernando Planella Gonzalez



Hoje **dados e informações** são as coisas mais
valiosas que tem.



Eugenio Facchini Neto

Conheça o livro da disciplina

CONHEÇA SEUS PROFESSORES 3

Conheça os professores da disciplina.

EMENTA DA DISCIPLINA 4

Veja a descrição da ementa da disciplina.

BIBLIOGRAFIA DA DISCIPLINA 5

Veja as referências principais de leitura da disciplina.

O QUE COMPÕE O MAPA DA AULA? 6

Confira como funciona o mapa da aula.

MAPA DA AULA 7

Veja as principais ideias e ensinamentos vistos ao longo da aula.

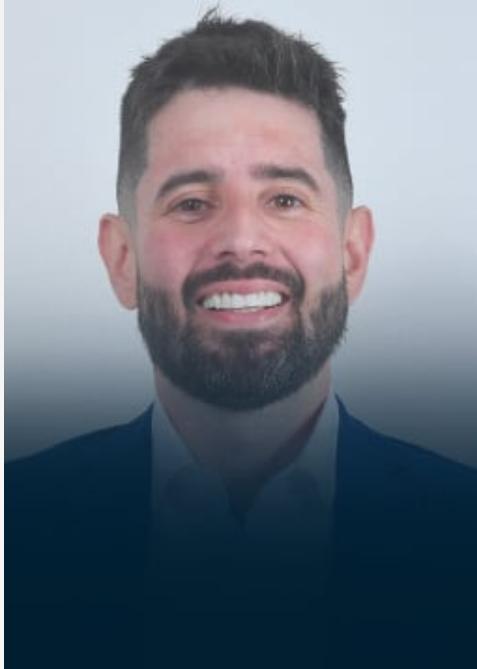
RESUMO DA DISCIPLINA 36

Relembre os principais conceitos da disciplina.

AVALIAÇÃO 37

Veja as informações sobre o teste da disciplina.

Conheça seus professores



VINICIUS SOARES

Professor Convidado

Entusiasta da Computação Distribuída, Vinicius Soares é Head de Tecnologia em uma das principais empresas do sul do país, ajudando clientes e empresas a alcançarem seus resultados de forma rápida e assertiva. Apaixonado por Java, Arquitetura de Sistemas e Computação em Nuvem, Vinicius possui sólida experiência liderando equipes de Arquitetura usando SOA e Microsserviços com tecnologias Open-sources. Compartilha suas experiências através de conteúdo online e eventos nacionais e internacionais como Devoxx, TDC e Campus Party. Como empreendedor, já ajudou mais de 1000 pessoas a se qualificarem para o mercado de TI e atuarem de forma representativa na área.

LUIS FERNANDO PLANELLA GONZALEZ

Professor PUCRS

Doutor Ciências da Computação (PUCRS, 2018). Desenvolvedor e arquiteto Java com experiência profissional desde 1999, certificado pela Sun como programador e desenvolvedor de componentes web na plataforma Java. Entusiasta de software livre.



Ementa da Disciplina

Estudo sobre a arquitetura de microserviços. Estudo sobre os conceitos de particionamento de serviços, replicação e distribuição, comunicação assíncrona via filas e Soluções serveless.

Bibliografia da Disciplina

As publicações destacadas têm acesso gratuito.

Bibliografia básica

MONTEIRO, Eduarda Rodrigues, et al. DevOps. Porto Alegre: Sagah, 2021.

GHIYA, Parth. TypeScript Microservices. Birmingham: Packt, 2018.

BENEVIDES, Rafael; POSTA, Christian. Microservices for Java Developers. Sebastopol: O'Reilly Media. 2019.

Bibliografia complementar

HASSAN, Sara; BAHSOON, Rami; KAZMAN, Rick. Microservice transition and its granularity problem: A systematic mapping study. Journal of Software: Practice and Experience. Volumen 50, Issue 9. Setembr, 2020.

FREEMAN, E. DevOps Para Leigos. Rio de Janeiro: Editora Alta Books, 2021.

CASALICCHIO, Emiliano; IANNUCCI, Stefano.The state-of-the-art in container technologies: Application, orchestration and security. Concurrency and Camputation - Pratice and Experience. Volume 32.

ESCOFFIER, Clement. Building Reactive Microservices in Java: Asynchronous and Event-Based Application Design. Sebastopol: O'Reilly Media. 2017.

YANAGI, Edson. Migrating to Microservice Databases: From Relational Monolith to Distributed Data. Sebastopol: O'Reilly Media. 2017.

RED HAT DEVELOPER. Developing microservices on Kubernetes. [S.I.]: Red Hat, [2021].

O que compõe o Mapa da Aula?

MAPA DA AULA

São os capítulos da aula, demarcam momentos importantes da disciplina, servindo como o norte para o seu aprendizado.



EXERCÍCIOS DE FIXAÇÃO

Questões objetivas que buscam reforçar pontos centrais da disciplina, aproximando você do conteúdo de forma prática e exercitando a reflexão sobre os temas discutidos. Na versão online, você pode clicar nas alternativas.



PALAVRAS-CHAVE

Conceituação de termos técnicos, expressões, siglas e palavras específicas do campo da disciplina citados durante a videoaula.



VÍDEOS

Assista novamente aos conteúdos expostos pelos professores em vídeo. Aqui você também poderá encontrar vídeos mencionados em sala de aula.



PERSONALIDADES

Apresentação de figuras públicas e profissionais de referência mencionados pelo(a) professor(a).



LEITURAS INDICADAS

A jornada de aprendizagem não termina ao fim de uma disciplina. Ela segue até onde a sua curiosidade alcança. Aqui você encontra uma lista de indicações de leitura. São artigos e livros sobre temas abordados em aula.



FUNDAMENTOS

Conteúdos essenciais sem os quais você pode ter dificuldade em compreender a matéria. Especialmente importante para alunos de outras áreas, ou que precisam relembrar assuntos e conceitos. Se você estiver por dentro dos conceitos básicos dessa disciplina, pode tranquilamente pular os fundamentos.

CURIOSIDADES

Fatos e informações que dizem respeito a conteúdos da disciplina.



DESTAQUES

Frases dos professores que resumem sua visão sobre um assunto ou situação.



ENTRETENIMENTO

Inserções de conteúdos para tornar a sua experiência mais agradável e significar o conhecimento da aula.



CASE

Neste item, você relembra o case analisado em aula pelo professor.



MOMENTO DINÂMICA

Aqui você encontra a descrição detalhada da dinâmica realizada pelo professor.



Mapa da Aula

Os tempos marcam os principais momentos das videoaulas.

AULA 1 • PARTE 1

SOA

O SOA representa a arquitetura orientada a serviços. O SOA não é:

- Não é uma tecnologia;
- Não é um produto;
- Não é um WebService;
- Não é um software;
- Não é um framework;
- Não é uma metodologia;
- Não é uma solução de negócio.

O SOA é uma abordagem arquitetural corporativa que permite a criação de serviços de negócios interoperáveis que podem ser facilmente reutilizados e compartilhados entre aplicações e empresas. O SOA é uma arquitetura baseada em reusabilidade, com serviços bem definidos e providos por componentes de TI. Os seus componentes possuem baixo acoplamento, provendo plataforma, tecnologia e linguagens independentes.

O SOA é composto por 4 pilares que trazem o alicerce da arquitetura: processos alinhados com as operações de negócio, processo de desenvolvimento, processo de deploy e todos os processos que envolvem o desenvolvimento daquele componente alinhado com o negócio. Alguns dos benefícios do SOA, são:

Desacoplamento: integrações inteligentes, flexibilidade, alinhamento com negócio;

Reutilização de serviços: produtividade, manutenibilidade;

Infraestrutura de plataforma: padronização corporativo (log, governança, entre outros).

02:57



06:00

PALAVRA-CHAVE

Reusabilidade: É a capacidade que um software tem de ser usado em novas aplicações.

06:47



“ Eu posso ter o serviço escrito em uma linguagem, feito em uma plataforma e uma empresa que trabalha com outra linguagem pode consumir esse serviço da mesma forma. ”

08:23



“ Tudo é baseado em pessoas quando falamos de computação distribuída. ”

13:42



Monólito x Microservices - Parte I

Uma arquitetura monolítica típica de um sistema complexo pode ser representada de forma que todas as funções do negócio estão implementadas em um único processo. A arquitetura monolítica é fácil de desenvolver, de fácil manutenção, necessita apenas de um deploy e possui tráfego de rede baixo, porém essa arquitetura possui alguns problemas:

“ Quando você precisa aumentar a performance do seu produto, a performance do seu sistema, você precisa aumentar o seu hardware. **”**

18:54

- Aumento de complexidade e tamanho ao longo do tempo;
- Alta dependência de componentes de código;
- Escalabilidade do sistema é limitada;
- Falta de flexibilidade;
- Dificuldade para colocar alterações em produção.

PALAVRA-CHAVE

API Gateway: É um gerenciador de tráfego que faz interface com o serviço de back-end real ou de dados e aplica políticas, autenticação e controle de acesso geral para chamadas de APIs de forma a proteger dados valiosos.



26:40

Uma das grandes desvantagens da arquitetura monolítica é que é muito difícil escalar o sistema. É muito comum que, a cada cliente que você tiver, você vá precisar replicar/duplicar o servidor e, consequentemente, isso duplica o seu custo. Quanto mais clientes tiver, mais servidor será necessário e mais caro ficará.

O microserviço é um serviço com um único propósito e que executa bem a sua tarefa dentro de um nível de granularidade, suportando as mudanças dos sistemas que são consideradas importantes tanto em tempo de projeto quanto em tempo de execução. O foco principal é tentar construir um software que possa se adaptar e isto só é possível de ser feito se as partes forem pequenas suficientes para se ajustar às diferenças nas mudanças de sua arquitetura. O microserviço tem como vantagens: a manutenção e evolução dos serviços mais estáveis, serviços com baixo nível de acoplamento e interdependência, escalabilidade do sistema, redução de custos, flexibilidade de tecnologia, facilidade de colocar alterações em produção, a resiliência, o aumento da produtividade, a implementação de entrega contínua, o monitoramento e automação de processos e, o foco na entrega de valor ao cliente.

EXERCÍCIO DE FIXAÇÃO

Qual das opções não é considerada benefício da arquitetura SOA:



AULA 1 • PARTE 2

Monólito x Microservices - Parte II

O professor dá continuidade ao conteúdo proposto anteriormente, abordando a diferença entre monólito e microserviço.

O trabalho com microserviços pode gerar riscos porque o risco aumenta a complexidade da coordenação, afetando a comunicação entre os microserviços.

PALAVRA-CHAVE

API RESTful: É uma interface de programação de aplicações que está em conformidade com as restrições do estilo de arquitetura REST, permitindo a interação com serviços web RESTful.

PALAVRA-CHAVE

Two pizzas team: Criado por Jeff Bezos, fundador da Amazon, o conceito é baseado na lógica de que se uma equipe que não pode ser alimentada por 2 pizzas, ela é grande demais para discutir ideias.

PALAVRA-CHAVE

Ciclo DevOps: A metodologia descreve abordagens que ajudam a acelerar os processos necessários para levar uma ideia do desenvolvimento à implantação em um ambiente de produção no qual ela seja capaz de gerar valor para o usuário. Essas ideias podem ser um novo recurso de software, uma solicitação de aprimoramento ou uma correção de bug, entre outros.



00:25



01:47

Características



04:15

O professor apresenta uma linha do tempo para demonstrar todas as etapas percorridas, desde a criação dos microserviços, passando por sua implementação em diversas empresas até chegar às inúmeras ferramentas que aderiram ao trabalho com microserviços. A partir de 2017, reatividade, governança e resiliência surgiram através de iniciativa da Netflix, que visava melhorar sua performance e que os seus serviços atendessem os seus usuários, conseguindo elevar o seu patamar. De acordo com Sam Newman, os microserviços são um conjunto de pequenos autônomos que trabalham juntos. Microserviços é um software modularizado em pequenos serviços que se comunicam através de uma forma padronizada e através de uma API RESTful (HTTP/Json).



09:36

Como características técnicas, os microserviços envolvem:



11:17

Out-of-process: possibilidade de execução fora dos processos.
Chamadas remotas: microserviços são acessados por chamadas remotas.

Independente de linguagem de programação: são agnósticos a linguagem de programação, ou seja, você pode ter serviços escritos em node, java, python, entre outros;

Baixo acoplamento: você é dono somente do seu domínio de negócio. Não necessitando de outro serviço para gerar novas versões e ou evoluir seu produto;

Práticas recomendadas

- Modele os serviços em torno de domínio da empresa;
- Descentralize tudo. Equipes individuais são responsáveis por projetar e criar serviços;
- Evite compartilhar esquemas de dados ou códigos;
- O armazenamento de dados deve ser privado para o serviço que é o proprietário dos dados. Use o melhor armazenamento para cada serviço e tipo de dados;
- Os serviços comunicam-se por meio de APIs bem projetadas. Evite o vazamento de detalhes da implementação. As APIs devem modelar o domínio, não a implementação interna do serviço;
- Evite acoplamento entre serviços. Causas de acoplamento incluem protocolos de comunicação rígidos e esquemas de banco de dados compartilhados;
- Descarregue preocupações abrangentes, como autenticação e terminação SSL, para o gateway. Mantenha o conhecimento de domínio fora do gateway;
- Os serviços devem ter um acoplamento flexível e alta coesão funcional. Isole falhas. Use estratégias de resiliência para impedir que falhas em um serviço distribuam-se em cascata.

13:18



Escalabilidade horizontal e vertical:

você pode aumentar o número de réplicas (scale horizontal) e/ou aumentar a capacidade computacional de seu serviço (scale vertical).

Como características organizacionais, podemos citar:

Agilidade: trabalhando em conjuntos de negócio menores você garante agilidade no desenvolvimento de software;

Equipe pequena e focada: utilizando o conceito de two-pizzas team você garante foco e produtividade do time de desenvolvimento;

Entregas rápidas: por ser altamente testável, um micro serviço pode estar disponível rapidamente em produção com garantias de segurança e qualidade;

Combinação de tecnologias: times podem ser multidisciplinares em tecnologias para desenvolver os microsserviços.

14:37



Cada microserviço possui o seu banco de dados, o banco de dados isolado.

16:15



PALAVRA-CHAVE

LGPD: A Lei Geral de Proteção de Dados tem como principal objetivo proteger os direitos fundamentais de liberdade e de privacidade e o livre desenvolvimento da personalidade da pessoa natural.

IaaS

A infraestrutura como serviço é um tipo de serviço de computação em nuvem que oferece recursos fundamentais de computação, armazenamento e rede sob demanda e pagos conforme o uso. A IaaS é um dos quatro tipos de serviços de nuvem, juntamente com o SaaS (software como serviço), a PaaS (plataforma como serviço) e a tecnologia sem servidor.

A IaaS tem com características:

- Computação de infraestrutura sob demanda;
- É um dos 3 modelos de serviços da computação em nuvem;

A IaaS provê:

- Servidores: computação e máquinas;
- Storage;
- Rede;
- Sistemas Operacionais.

O usuário, ao invés de adquirir softwares ou máquinas, espaço em data centers ou equipamentos de rede, praticamente aluga espaços para estes recursos em uma infraestrutura externa.

IaaS pode ser obtida em:

Nuvem pública: é considerada nuvem pública uma infraestrutura que consiste de recursos compartilhados, liberados sob demanda, baseado na internet;

Nuvem privada: incorpora a maioria das features de uma nuvem pública como virtualização porém fica dentro de uma rede privada;

Nuvem híbrida: mistura de uma nuvem privada com uma nuvem pública, geralmente conectadas através de um túnel VPN.

25:22

30:27

Quando utilizar e quando não utilizar

Devemos utilizar quando a demanda for volátil, ou seja, você tem a possibilidade de diminuir e aumentar sua capacidade computacional de acordo com a necessidade. Empresas sem capacidade de investimento em hardware, empresas com crescimento rápido e necessidade de escala rápida de sua infraestrutura.

Não devemos utilizar quando a legislação não permitir guardar os dados fora da infraestrutura interna da empresa, ou a terceirização do armazenamento não é permitida. Não é aconselhável quando os níveis de desempenho necessários para aplicação tenham limite de acesso ao provedor da nuvem.

34:28

PaaS - Parte I

A plataforma como serviço é um ambiente de desenvolvimento e implantação completo na nuvem, com recursos que permitem a você fornecer tudo, desde aplicativos simples baseados em nuvem até sofisticados aplicativos empresariais habilitados para a nuvem.

A PaaS tem como características:

Pode ser considerada IaaS adicionada uma camada middleware e/ou componentes prontos;

Uma camada de abstração entre seu aplicativo em nuvem e seu provedor de IaaS;

É um ambiente de execução escalável e com alta disponibilidade para aplicações customizadas;

É uma categoria de computação em nuvem que fornece uma plataforma e um ambiente para permitir que os desenvolvedores criem aplicativos e serviços pela Internet;

Fornece fundamentalmente escala elástica do seu aplicativo.

AULA 1 • PARTE 3

PaaS - Parte II

Alguns dos benefícios da PaaS são:

- Infraestrutura na nuvem, escalável e com alta disponibilidade nativa;
- Alta produtividade no desenvolvimento e manutenção de aplicações sob demanda;
- Resumindo: baixo custo (TCO), confiabilidade e diminuição do tempo de entrega.

A PaaS também tem como vantagens:

Desenvolvimento 100% focado no negócio: por direcionar a arquitetura lógica e administrar a arquitetura física de forma bem transparente, é possível desenvolver uma aplicação que vá demandar uma requisição por minuto ou 10 mil requisições por segundo da mesma forma, com o mesmo nível de preocupação do ponto de vista técnico: apenas a lógica de negócios;

Produtividade: o simples fato de não se gerenciar balanceamento de carga, replicação, cluster, instalando e configurando middlewares (servidores de aplicação, banco de dados, etc.) já é um grande ganho. Além disso, os grandes fornecedores estão criando camadas de componentes prontos para uso, APIs e aceleradores de desenvolvimento nessas plataformas.

Existe esforço dos fornecedores para deixar essa camada o mais padrão possível, mas ainda existe uma boa parte que é proprietária. Ao adotar PaaS, é natural que se adote também essa camada proprietária, caso contrário, poderia se trabalhar direto na infraestrutura. É esta camada que permite dar um salto de produtividade, lidando com escalabilidade e disponibilidade de forma transparente. A decisão a ser tomada é: menos custo e mais entregas contra o efeito “lock-in” das aplicações construídas nessa abordagem.

05:28



00:25



PALAVRA-CHAVE

Lock-in: É usado para se referir a um cenário em que você adquire uma solução tecnológica e depois se vê impossibilitado de trocá-la por outra, estando trancado dentro dela. É comum que haja um profissional dedicado apenas a estudar as soluções tecnológicas do mercado.

07:59



The Twelve - Factor For App

A metodologia fornece doze práticas definidas para permitir que aplicações possam ser construídas visando portabilidade e resiliência quando implantadas na web. São elas:

Base de código:

- Somente uma base de código por aplicação;
- Vários deploys por aplicação;
- O desenvolvedor possui uma cópia local do repositório.

Dependências:

- Declare e isole explicitamente as dependências;
- Uma aplicação 12 fatores nunca confia na existência implícita de pacotes em todo o sistema;
- Uma declaração de dependência explícita é que simplifica a configuração da aplicação para novos desenvolvedores;
- Na prática: tenha sempre um gerenciador de dependências configurado para seu projeto (maven, gradle, npm, pip e etc).

“ Uma declaração explícita simplifica a configuração para os nossos desenvolvedores. **”**

“

10:53

“ A troca de ambiente não pode afetar a sua aplicação. **”**

“

13:16

“ A configuração distribuída tem que ser planejada desde o início da concepção da sua plataforma de microsserviço. **”**

“

13:36

“ Não existe microsserviço stateful, microsserviço é sempre stateless. **”**

“

18:13

PALAVRA-CHAVE

Container: Tem o objetivo de segregar e facilitar a portabilidade de aplicações em diferentes ambientes. Um container contém um conjunto de processos que são executados a partir de uma imagem. Os containers isolam os processos da aplicação do restante do sistema operacional.

🔍

21:00

Configurações:

- A configuração de uma aplicação é tudo que é provável variar entre deploys (homologação, produção, desenvolvimento, etc).
- Uma aplicação 12 fatores armazena configuração em variáveis de ambiente ou algum recurso de configuração distribuída.
- Necessitamos de facilidade na troca de ambientes sem ter a necessidade de alterar o codebase.

Serviços de apoio:

- Trate serviços de apoio como recursos anexados;
- Serviço de apoio é qualquer serviço que o App consuma via rede como parte de sua operação normal;
- Não se deve fazer distinção entre serviços locais e terceiros.

Construa, lance e execute:

- Uma base de código é transformada em um deploy através de 3 estágios:
- Construção: converte o repositório em um pacote executável.
- Lançamento: combina o artefato construído com a configuração do deploy.
- Execução: roda o app no ambiente de execução através dos processos específicos do APP.
- O app 12 fatores utiliza separação estrita entre os estágios de construção, lançamento e execução.

Processos:

- Você não deve introduzir estado em seus serviços; os aplicativos devem ser executados como um processo único e sem estado;
- Os processos dos Doze Fatores são Stateless e não compartilham nada. Esse fator está no núcleo da arquitetura de microsserviços.

Aplicações - Cloud Native

É uma abordagem moderna para construir e executar aplicativos de software que exploram a flexibilidade, escalabilidade e resiliência da computação em nuvem.

As aplicações englobam as várias ferramentas e técnicas usadas pelos desenvolvedores de software hoje para construir aplicativos para a nuvem pública, em oposição às arquiteturas tradicionais adequadas para um data center local.

Definição CNCF:

- Containerizado;
- Gerenciado dinamicamente.
- Automação;
- Registro e descoberta;
- Rastreamento distribuído/observabilidade.

Cloud:

- Elasticidade;
- Modelo on-demand.

Definição Pivotal:

- Processos;
- Ferramentas;
- Cultura.

Entrega contínua:

- Automação.

Microsserviços:

- Automação;
- Registro e Descoberta;
- Rastreamento distribuído/observabilidade;
- Anti-fragilidade / Engenharia de caos.



27:33

Vínculo de porta:

- Seu serviço deve estar visível para outras pessoas via ligação de alguma porta. Se você criou um serviço, verifique se outros serviços podem tratar isso como um recurso. O aplicativo de doze fatores é completamente independente de outros recursos.

Concorrência:

- Divilde seu aplicativo em pequenos pedaços, em vez de tentar aumentar seu aplicativo (executando uma única instância na máquina mais poderosa disponível). Aplicativos pequenos e definidos permitem a expansão conforme necessário para lidar com cargas variadas. O processo deve ser dimensionado individualmente, com o Fator 6 (sem estado), torna-se transparente este tipo de abordagem.

Descartabilidade:

- Os processos devem consumir menos tempo. Certifique-se de poder correr e parar rapidamente. E que você pode lidar com falhas. Sem isso, o dimensionamento automático e a facilidade de implantação e desenvolvimento estão sendo diminuídos. Você pode conseguir isso com contêineres.

Paridade entre desenvolvimento e produção:

- Mantenha o desenvolvimento, a homologação e a produção o mais semelhante possível, para que qualquer pessoa possa utilizá-lo da mesma forma. A implantação contínua precisa de integração contínua com base em ambientes correspondentes para limitar desvios e erros. Isso também incentiva implicitamente uma cultura DevOps na qual o desenvolvimento e as operações de software são unificados.

BOSH:

- Suporte a múltiplas clouds;
- Separação clara entre sistemas;
- Provisionamento rápido;
- Escalabilidade;
- Monitoramento de saúde;
- Controle de falhas;
- Deploy canário.

EXERCÍCIO DE FIXAÇÃO

Assinale a alternativa incorreta correspondente aos problemas da arquitetura monolítica:



Logs:

- Trate os logs como fluxos de eventos. O registro é importante para validar erros e também verificar a integridade geral do seu sistema. Ao mesmo tempo, seu aplicativo não deve se preocupar com o armazenamento dessas informações. Esses logs devem ser tratados como um fluxo contínuo capturado e armazenado por um serviço separado.

Processos administrativos:

- Execute tarefas administrativas / gerenciamento como processos pontuais - tarefas como migração de banco de dados ou execução de scripts pontuais no ambiente. Para evitar mexer com o banco de dados, use as ferramentas criadas ao lado do aplicativo e isole completamente sua aplicação, por exemplo.
- Alguns pontos nos 12 fatores podem parecer triviais, mas ao executar mais de 20 serviços em poucos ambientes, eles podem ser de grande importância.

AULA 1 • PARTE 4

PALAVRA-CHAVE



00:43

Kubernetes: É um sistema de orquestração de contêineres open-source que automatiza a implantação, o dimensionamento e a gestão de aplicações em contêineres.

Containers

Os containers são métodos utilizados na implementação e na execução de aplicativos distribuídos sem que haja a necessidade de configuração de uma máquina virtual para cada um desses aplicativos. Container é um pequeno sistema Linux minimalista. Ele compartilha o Kernel do Host e seus processos são trabalhados isoladamente. Os containers possuem Commits e versionamento de containers, compartilhando através de ambientes customizados.

Por que os contêineres são bons para microserviços?

Projetado para executar um aplicativo por contêiner;

Separação natural da carga de trabalho;

Muito leve;

Ótimo para dimensionar rapidamente;

Melhor uso de recursos;

Os contêineres compartilham o SO host e, quando apropriado, binários e bibliotecas;

Formatos padrão de contêiner, como o Docker, são distribuições cruzadas entre linux compatível;

Incrivelmente fácil de mover sua carga de trabalho

Equilibre melhor os recursos do sistema;

Permita que os desenvolvedores trabalhem em um ambiente de produção simulado;

Remove o problema “funcionou na minha máquina”.

01:20

07:11

08:59

Características cloud native

Arquiteturas cloud native aprimoram nossa capacidade de praticar DevOps e Entrega Contínua (Continuous Delivery), e elas exploram as características da infraestrutura na nuvem (Cloud Infrastructure). “Cloud-native” é um adjetivo que descreve as aplicações, arquiteturas, plataformas/infraestrutura, e processos que, em conjunto, tornam “econômico” trabalhar de forma a melhorar nossa capacidade de responder rapidamente às mudanças e reduzir a imprevisibilidade. As arquiteturas cloud-native aprimoram nossa capacidade de praticar DevOps e Continuous Delivery, e elas exploram as características da infraestrutura na nuvem. As arquiteturas cloud-native são definidas tendo as seguintes seis qualidades:

- Modularidade (através de Microservices);
- Capacidade de observação;
- Implementabilidade;
- Testabilidade;
- Descartabilidade;
- Substituível.

Uma aplicação Cloud-native é criada para ser:

- Escalável;
- Tolerante a falhas;
- Decomposto em serviços;
- Envia o máximo trabalho para a plataforma se possível;
- Automatizado.

“ O microserviço por standard, por default, é projetado para executar isoladamente. E um container é projetado para executar um aplicativo somente. **”**

Docker

É uma plataforma aberta, criada com o objetivo de facilitar o desenvolvimento, a implantação e a execução de aplicações em ambientes isolados. Foi desenhada especialmente para disponibilizar uma aplicação da forma mais ágil. Tecnologia Open Source que permite criar, executar, testar e implantar aplicações distribuídas dentro de containers de software.

Ele permite que você empacote um software de uma padronizada para o desenvolvimento de software, contendo tudo que é necessário para a execução: código, runtime, ferramentas, bibliotecas, etc.

Docker permite que você implante aplicações rapidamente, de modo confiável e estável, em qualquer ambiente. Existem mais de 500 mil aplicações Dockerizadas, um crescimento de 3100% ao longo de 2 anos e mais de 4 bilhões de containers já foram puxados até hoje. O Docker é apoiado por uma grande e crescente comunidade de colaboradores e usuários. A adoção do Docker aumentou mais de 30% no último ano e em cerca de 30% dos containers Docker estão rodando em produção. 29% das empresas que já ouviram falar em Docker planejam usá-lo.

A parte cliente fala com o Docker daemon, que faz o trabalho pesado de construção, execução e distribuição de seus containers e imagens Docker, também controla os recursos executados. O cliente Docker e Docker daemon, podem ser executados no mesmo sistema, também é possível conectar um cliente Docker a um Docker daemon remoto. O cliente Docker e daemon se comunicam através de uma API REST, através de sockets UNIX ou uma interface de rede, para execuções de comandos ou scripts.

14:54

20:15

Características das arquiteturas

Containers tem como base sempre uma imagem, pense como na seguinte analogia do mundo Java, uma imagem é uma classe e um container é como um objeto instância dessa classe, então podemos através de uma imagem “instanciar” vários containers, também através de recursos chroot, Cgroups é possível definirmos limitações de recursos recursos e isolamento parcial ou total dos mesmos.

Algumas características dos containers:

- Portabilidade de aplicação;
- Isolamento de processos;
- Prevenção de violação externa;
- Gerenciamento de consumo de recursos.

Imagens são templates para criação de containers, como falado no slide anterior, imagens são imutáveis, para executá-las é necessário criar uma instância dela (container), também vale ressaltar que as imagens são construídas em camadas, o que facilita sua reutilização e manutenção. Em resumo, uma imagem nada mais é do que um ambiente totalmente encapsulado e pronto para ser replicado onde desejar.

Dockerfile são scripts com uma série de comandos para criação de uma imagem, nesses scripts podemos fazer uma série de coisas como executar comandos sh, criar variáveis de ambiente, copiar arquivos e pastas do host para dentro da imagem.

Docker registry é como um repositório GIT, onde as imagens podem ser versionadas, comitadas, “puxadas” etc, quando recuperamos uma imagem, usando o comando docker pull por exemplo, estamos normalmente baixando a imagem de um registro Docker, o repositório oficial do Docker é o Docker HUB, onde é possível hospedar e versionar imagens públicas e privadas.

PALAVRA-CHAVE

Java 17: É o lançamento de suporte de longo prazo (LTS) sob a cadência de lançamento de seis meses do Java e é o resultado de uma ampla colaboração entre engenheiros da Oracle e outros membros da comunidade mundial de desenvolvedores Java.



23:23



EXERCÍCIO DE FIXAÇÃO

Alguns dos benefícios do PaaS, são:

AULA 2 • PARTE 1

Maturidade com Cloud Native

Cloud Native é muito novo e a maioria das empresas não têm experiência para navegar por conta própria. É muito fácil e caro cometer erros. A matriz de maturidade de soluções de cloud avalia a disponibilidade da nuvem em nove áreas cruciais de transformação, tanto técnica quanto centrada no ser humano. Os resultados definem e descrevem onde você está agora e mostram para onde ir.

As empresas podem gerar enorme crescimento econômico e valor comercial, fornecendo serviços ou aplicativos baseados na nuvem: instagram, uber, whatsapp e ainda pequenas empresas surpreendentemente (se relacionarmos o número modesto de funcionários dessas empresas em seu crescimento econômico notável) cujos serviços são usados e distribuídos com frequência com cloud computing.



01:44

06:03



Empresas que nascem na cloud, empresas que já dominam como utilizar a cloud, tendem a ter um sucesso muito maior porque eles não vão ter o custo.

No entanto, mesmo um modelo de negócios de crescimento rápido deve ter consequências e dependências a longo prazo, por isso maturidade em cloud é fator principal para escalar negócios, arquiteturas e usuários. Segundo o Open Data Center Alliance, possuímos alguns passos para fazer nossa passagem e maturidade num modelo de maturidade de nuvem. Os tipos de maturidade são definidos como:

Cloud ready (nível 0): operado em infraestrutura virtualizada. Criação das instâncias ou imagem por script;

Cloud friendly (nível 1): composto por serviços vagamente acoplados, os serviços podem ser descobertos pelo nome. Os componentes são projetados para padrões de nuvem. A computação e o armazenamento são separados;

Cloud resilient (nível 2): estado é isolado em um mínimo de serviços, não afetado por falhas de serviço dependentes. A infraestrutura é independente;

Cloud native (nível 3): transferível entre provedores de infraestrutura em tempo de execução e sem interrupção do serviço. dimensionar dentro/frente automaticamente com base em estímulos de requisições e volumetria.

Domain drive design (modelo) - Parte I

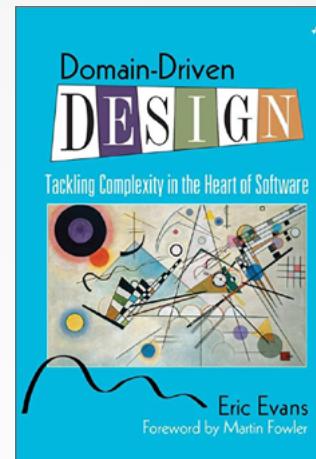
O modelo é evolutivo: a cada interação entre especialistas de domínio e a equipe técnica, o modelo se torna mais profundo e expressivo, mais rico e os desenvolvedores transferem essa fonte de valor para o software. Assim, o modelo vai sendo gradualmente enriquecido com o expertise dos especialistas do domínio destilado pelos desenvolvedores, fazendo com que o time ganhe cada vez mais insight sobre o negócio e que esse conhecimento seja transferido para o modelo (para o código) através dos blocos de construção do DDD.

18:27



LEITURA INDICADA

Livro: Domain-Driven Design: Tackling Complexity in the Heart of Software



Reunindo práticas de design e implementação, este livro incorpora vários exemplos baseados em projetos que ilustram a aplicação do design dirigido por domínios no desenvolvimento de softwares na vida real.

18:41



Domain drive design

Fundamentado na experiência de mais de 20 anos de Eric Evans no desenvolvimento de sistemas, o DDD é uma abordagem que reúne um conjunto de boas práticas, padrões, ferramentas e recursos de orientação a objetos que têm como objetivo a construção e desenvolvimento de sistemas de acordo com o domínio e regras de negócio do cliente. Além disso, questões relacionadas ao processo de desenvolvimento, como a necessidade de um estreito relacionamento entre a equipe de programadores e os especialistas do domínio, também são tratadas pela abordagem.

29:17



O principal conceito do DDD é o modelo. O modelo expressa o domínio e negócio do cliente e pode ser criado utilizando desenhos, fluxogramas, diagramas, entre outros. O importante é que ele represente o negócio do cliente. Como principais componentes do DDD, podemos listar: a linguagem onipresente, a arquitetura em camadas e os padrões.

EXERCÍCIO DE FIXAÇÃO

Qual é o principal conceito da abordagem Domain drive design?



É uma abordagem de desenvolvimento de software que reúne um conjunto de conceitos, princípios e técnicas cujo foco está no domínio e na lógica do domínio com o objetivo de criar um Domain Model. Significa desenvolver software de acordo com o domínio relacionado ao problema que estamos propondo resolver. O foco da abordagem é criar um domínio que fale a língua do usuário usando o que é conhecido como linguagem Ubíqua.

Em linguagem Ubíqua entende-se que ao trabalhar com DDD devemos conversar usando uma mesma língua, em um único modelo, de forma que o mesmo seja compreendido pelo cliente, analista, projetista, entre outros, nesta linguagem, que seria a linguagem usada no dia a dia.

AULA 2 • PARTE 2

Domain drive design (modelo) - Parte II

O professor dá continuidade ao conteúdo proposto anteriormente. Quando novas regras de negócio são adicionadas e/ou regras existentes são alteradas ou removidas, a implementação é refatorada para refletir essas alterações do modelo no código. No final, o modelo (que em última instância será o software) vai expressar com riqueza de conhecimento o negócio. Algumas das vantagens, são:

- O código fica menos acoplado e mais coeso;
- O negócio é melhor compreendido por todos da equipe o que facilita o desenvolvimento;
- Alinhamento do código com o negócio;
- Favorecer reutilização;
- Mínimo de acoplamento;
- Independência da tecnologia.



00:25

05:53



Domain drive design e Event Storming

Event Storming é uma técnica de design rápido que engaja especialistas do domínio de negócios com desenvolvedores para que alcancem um ciclo rápido de aprendizagem (aprender o máximo possível no menor tempo possível. Segundo Martin Fowler, Bounded Context ou (contexto limitado) é um padrão central no design orientado a domínio. É o foco da seção de design estratégico da DDD que trata de lidar com grandes modelos e equipes. O DDD lida com modelos grandes, dividindo-os em diferentes contextos limitados e sendo explícito sobre suas inter-relações. Para encontrar os comandos, agregações e boundaries usaremos o conceito de Evento Storming.

PERSONALIDADE

Martin Fowler



É um autor conhecido na área de arquitetura de software, especializado em análise orientada a objetos, UML, padrão de projeto de software e metodologias de desenvolvimento ágil de software, incluindo Programação Extrema.

Maturidade com Cloud Native

Nesta etapa, você muda da análise do domínio para os primeiros estágios do design do sistema. Até esse momento, você está simplesmente tentando entender como os eventos no domínio se relacionam - é por isso que a participação de especialistas em domínios é tão crítica. No entanto, para criar um sistema que implemente o processo de negócios em que você está interessado, é necessário passar à questão de como esses eventos ocorrem. Os comandos são o mecanismo mais comum pelo qual os eventos são criados. A chave para encontrar comandos é fazer a pergunta: "Por que esse evento ocorreu?". Nesta etapa, o foco do processo passa para a sequência de ações que levam a eventos. Seu objetivo é encontrar as causas pelas quais os eventos registram os efeitos. Outra parte importante do processo que se torna mais detalhada nessa etapa é a descrição de políticas que podem acionar a geração de um evento a partir de um evento anterior (ou conjunto de eventos). Avalie se o elemento de dados é uma entidade comercial principal, identificada exclusivamente por uma chave, suportada por vários comandos.



06:20

Event Storming é uma técnica de design rápido que engaja especialistas do domínio de negócios com desenvolvedores para que alcancem um ciclo rápido de aprendizagem (aprender o máximo possível no menor tempo possível). Event Storming permite:

- Mapear os eventos;
- Identificar os comandos;
- Associar os aggregates;
- Delimitar as fronteiras do modelo;
- Identificar os domínios de negócio.

08:34



Eventos de domínio

Use eventos de domínio para implementar explicitamente os efeitos colaterais de alterações em seu domínio. Em outras palavras, e usando terminologia DDD, use eventos de domínio para implementar explicitamente efeitos colaterais entre várias agregações. Opcionalmente, para melhor escalabilidade e menor impacto em bloqueios de banco de dados, use consistência eventual entre agregações dentro do mesmo domínio. Um evento é algo que ocorreu no passado. Um evento de domínio é algo que ocorreu no domínio que você deseja que outras partes do mesmo domínio (em processo) tenham conhecimento.



15:59

As partes notificadas geralmente reagem de alguma forma aos eventos. Um benefício importante dos eventos de domínio é que os efeitos colaterais podem ser expressos explicitamente. Em resumo, eventos de domínio ajudam você a expressar, explicitamente, as regras de domínio, com base na linguagem ubíqua fornecida pelos especialistas do domínio. Os eventos de domínio também permitem uma melhor separação de interesses entre classes dentro do mesmo domínio. É importante garantir que, assim como uma transação de banco de dados, todas as operações relacionadas a um evento de domínio sejam concluídas com êxito ou nenhuma delas seja.

Tem uma vida útil ao longo do processo de negócios. Isso levará ao desenvolvimento de uma análise do ciclo de vida da entidade. Esse primeiro nível de definição de dados ajuda a avaliar o escopo e a responsabilidade do micro serviço à medida que você começa a ver pontos em comum emergindo dos dados usados entre vários eventos relacionados.

- O comando de disparo é identificado em um post-it azul;
- O comando pode se tornar uma operação de micro serviço exposta via API;
- A pessoa humana que emite o comando é identificada e mostrada em uma nota laranja;
- Alguns eventos podem ser criados aplicando políticas de negócios;
- Identificação dos comandos que geram os eventos;
- Geralmente os comandos estão associados à alguma ação do usuário, interação com sistema externo ou gerados por um temporizador/cron;
- Verbo na forma imperativa;
- Deve ser colocado no lado esquerdo do evento que ele gera;
- Durante o processo, é comum identificar que um comando pode gerar vários eventos.

Agregações

Agregação é um padrão no Domain Driven Design. Um agregado DDD é um cluster de objetos de domínio que podem ser tratados como uma única unidade. Um exemplo pode ser um pedido e seus itens de linha, esses serão objetos separados, mas é útil tratar o pedido (junto com seus itens de linha) como um único agregado. Quaisquer referências externas ao agregado devem apenas ir para a raiz agregada. A raiz pode assim garantir a integridade do agregado como um todo.

Os eventos de domínio são parecidos com eventos do estilo de mensagens, com uma diferença importante. Com mensagens reais, enfileiramento de mensagens, agentes de mensagens ou um barramento de serviço que usa o AMQP, uma mensagem é sempre enviada de forma assíncrona e comunicação entre processos e computadores. Isso é útil para a integração de vários contextos delimitados, microsserviços ou até mesmo aplicativos diferentes. No entanto, com os eventos de domínio, ao acionar um evento na operação de domínio em execução no momento, você deseja que os efeitos colaterais ocorram dentro do mesmo domínio.

- A primeira etapa do Event Storming consiste em mapear os eventos que ocorrem no domínio que está sendo estudado;
- Um evento é qualquer coisa relevante que aconteceu no passado e tende a ser de simples compreensão para pessoas não técnicas;
- O padrão para descrever o evento é utilizar o verbo no passado e deve-se tentar mapear todos os eventos;
- Para essa etapa, utilizamos os post-its de cor laranja;
- Talvez possa existir um pouco de dificuldade inicial para explicar o que é um evento para os especialistas de negócios (não técnicos);
- O facilitador deve cuidar para que, sem que percebam, os participantes fujam do escopo da etapa e começem a citar o mapeamento das entidades ou regras de negócios;
- Alguns exemplos de eventos identificados são Produto Ativado, Produto Desativado, Licença Utilizada, Licença Liberada, Novo Ambiente Criado.



21:50

Agregados são o elemento básico da transferência de armazenamento de dados - você solicita carregar ou salvar agregados inteiros. As transações não devem cruzar fronteiras agregadas. Às vezes, os agregados DDD são confundidos com as classes de coleção (listas, mapas, etc.). Agregados DDD são conceitos de domínio (ordem, visita à clínica, lista de reprodução), enquanto as coleções são genéricas. Um agregado geralmente contém coleções múltiplas, junto com campos simples.

O termo “agregado” é comum e é usado em vários contextos diferentes (por exemplo, UML), caso em que não se refere ao mesmo conceito que um agregado DDD. Os aggregates são a parte do sistema que recebem os comandos e que geram os eventos, eles são os objetos que armazenam os dados e são modificados pelos comandos. Aplicativos tradicionais têm usado frequentemente transações de banco de dados para impor a consistência. Em um aplicativo distribuído, no entanto, isso muitas vezes não é viável.

Uma única transação empresarial pode abranger vários repositórios de dados, ser demorada ou envolver serviços de terceiros. Por fim, cabe ao aplicativo, não à camada de dados, impor as variáveis necessárias para o domínio. É isso que as agregações destinam-se a modelar. São a parte do sistema que recebem os comandos e que geram os eventos, eles são os objetos que armazenam os dados e são modificados pelos comandos. Pode-se utilizar o nome da entidade ou dado quando se fala sobre Aggregate. Durante o exercício, pode ser que os Aggregates se repitam ao longo da linha do tempo, mas não se deve agrupá-los.

PALAVRA-CHAVE

GRPC: Conhecido como Google Remote Procedure Call, a técnica é um sistema de código aberto de chamada de procedimento remoto desenvolvido inicialmente no Google em 2015.

23:19



Um agregado geralmente contém coleções múltiplas, junto com o campo simples.



23:35



Um agregado é a junção de vários ítems que vão compor o meu evento, o meu domínio.



25:54



Limites

Bounded Context é um padrão central no design orientado a domínio e é o foco da seção de design estratégico da DDD que trata de lidar com grandes modelos e equipes. O DDD lida com modelos grandes, dividindo-os em diferentes contextos limitados e sendo explícito sobre suas inter-relações. À medida que você tenta modelar um domínio maior, fica progressivamente mais difícil criar um único modelo unificado. Diferentes grupos de pessoas usarão vocabulários sutilmente diferentes em diferentes partes de uma grande organização.

A precisão da modelagem rapidamente se depara com isso, muitas vezes levando a muita confusão. Normalmente, essa confusão se concentra nos conceitos centrais do domínio. Contextos limitados têm conceitos não relacionados (como um tíquete de suporte existente apenas em um contexto de suporte ao cliente), mas também compartilham conceitos (como produtos e clientes). Contextos diferentes podem ter modelos completamente diferentes de conceitos comuns, com mecanismos para mapear entre esses conceitos polissêmicos para integração. Vários fatores traçam limites entre contextos.

34:07



EXERCÍCIO DE FIXAÇÃO

Quais os 3 pilares do Domain-Driven Design?



Normalmente, a dominante é a cultura humana, já que os modelos agem como linguagem onipresente, você precisa de um modelo diferente quando a linguagem muda. Você também encontra vários contextos no mesmo contexto de domínio, como a separação entre os modelos de banco de dados relacional e na memória em um único aplicativo. Esse limite é definido pela maneira diferente como representamos os modelos.

AULA 2 • PARTE 3

Comunicação entre serviços

Há duas formas para comunicação: síncrona e assíncrona. Na maneira síncrona há comunicação em tempo real. Balanceador de carga verifica o nível de infraestrutura e o tratamento de erros pode ser pelo status do http. A maneira assíncrona não espera resposta. A comunicação entre os dados pode ser “delay” entre as estruturas. O balanceador de carga pode ser uma fila. O tratamento de erros pode ficar no gerenciador da fila, em caso de erros o gerenciador de mensagens pode tratar o reenvio.



00:25

04:43



PALAVRA-CHAVE

Message broker: É um software que possibilita que aplicativos, sistemas e serviços se comuniquem e troquem informações. O message broker faz isso convertendo mensagens entre protocolos de mensagens formais.

06:46



PALAVRA-CHAVE

12:20



“ Apesar de tudo, a minha aplicação é a fonte de eventos. ”

Dispare e esqueça: O modo disparar e esquecer pode ser ativado somente quando os aplicativos usam um pool de conexão. Se um aplicativo ativar e esquecer para uma única conexão, a configuração do modo será ignorada e a inserção será executada em um único encadeamento.

“ Se o seu orquestrador está chamando mais de 8 microserviços, está na hora de rever. ”

19:38



PALAVRA-CHAVE

Orquestração: A orquestração ajuda a gerenciar fluxos de trabalho e tarefas complexas com mais facilidade. Ela é compatível com uma abordagem de DevOps e ajuda a sua equipe a implantar aplicações mais rapidamente.

PALAVRA-CHAVE

Saga: É uma maneira de gerenciar a consistência de dados entre microserviços em cenários de transação distribuída. Uma saga é uma sequência de transações que atualiza cada serviço e publica uma mensagem ou evento para disparar a próxima etapa de transação.

24:19



PALAVRA-CHAVE

Coreografia: É uma técnica para composição de serviços de forma distribuída e descentralizada, vista sob uma perspectiva global. A coreografia é o resultado do conhecimento e comportamento coletivo espalhado pelo sistema.

28:33



EXERCÍCIO DE FIXAÇÃO

O que define um domínio?



AULA 2 • PARTE 4

“ Não existe uma maneira de trabalhar com microserviços sem ter o registro deles. ”

Integração e deploy contínuo

A integração é uma prática de desenvolvimento de software onde os membros do time de desenvolvimento integram seu trabalho constantemente. Cada integração é feita automaticamente por um processo para detectar falhas rapidamente. Reduz drasticamente problemas de integração e possibilita o desenvolvimento de um software seguro e coeso. O professor explica pontos-chave sobre a integração:

- Escreva testes automatizados para desenvolvedores;
- Execute compilações privadas;
- Confirme código com frequência;
- Não confirme código com defeito;
- Evite obter código com defeito;
- Corrija construções com defeito imediatamente;
- Todos os testes e inspeções devem passar.

O deploy contínuo é muito semelhante à entrega contínua, porém dá um passo adiante em relação à automação. O deploy contínuo não precisa de uma área de teste para ser revisada manualmente, já que os testes que são automatizados e integrados no começo de todo o processo de desenvolvimento, seguindo todas as fases do release.

“ 01:35

05:02

16:07

19:12

01:35

06:40

06:40

06:40

PALAVRA-CHAVE

Server-side: Também conhecido por back-end, é um termo usado para designar operações que, em um contexto cliente-servidor, são feitas no servidor, não no cliente.

PALAVRA-CHAVE

Pirâmide de teste: É uma forma gráfica de demonstrar de maneira simples os tipos de testes, seus níveis, velocidade de implementação e complexidade dos testes realizados.

PERSONALIDADE

Jez Humble



É diretor na ThoughtWorks Studios e tem trabalhado com uma variedade de plataformas e tecnologias, dando consultoria para organizações sem fins lucrativos, empresas financeiras e de telecom, e companhias de varejo online.

Build Pipeline

É uma implementação do paradigma contínuo, em que builds, testes e implementações automatizadas são orquestrados como um único fluxo de trabalho de lançamento. Alguns fatores da etapa de **confirmação**:

- Poll SCM;
- Compilação;
- Cobertura dos testes;
- Unitários e integração;
- Qualidade do código fonte;
- Conformidades com padrões empresariais;
- Possíveis bugs;
- Boas práticas;
- Empacotamento e publicação.

Estágio de **aceitação**:

- Testes automatizados;
- Testes funcionais;
- Testes de aceitação;
- Testes de performance;
- Testes de segurança.

Automatização atrelada com teste gera sucesso, reduzindo os riscos com releases incrementais. Existem inúmeras formas de se construir um build pipeline, descubra a sua. A cultura de DevOps é necessária.

Algumas das **dificuldades** citadas, são:

- Exige mudança cultural;
- Maior dificuldade encontrada nas implantações de continuous delivery;
- Exige ciclos curtos;
- Empresas organizadas em silos/áreas;
- Requer envolvimento de todos.



22:00



24:41



Você não vai conseguir trabalhar com deploy contínuo, integração contínua sem uma cultura DevOps.



29:22

PALAVRA-CHAVE

Rollback: As reversões são importantes para a integridade do banco de dados, pois significam que o banco de dados pode ser restaurado para uma cópia limpa mesmo após a execução de operações incorretas.



33:24

PALAVRA-CHAVE

Flyway: É uma ferramenta que permite o versionamento e gerenciamento do Banco de dados. Com ele podemos controlar a evolução dos elementos que compõem uma determinada base de dados, sendo eles tabelas, sequences, views, entre outros.



EXERCÍCIO DE FIXAÇÃO

Sobre as comunicações síncrona e assíncrona, é incorreto afirmar que:

AULA 3 • PARTE 1

Comunicação assíncrona via filas

O professor inicia sua aula fazendo uma breve revisão sobre os conteúdos abordados anteriormente. O modelo mais básico de comunicação entre processos é o modelo de requisição/resposta, também conhecido como cliente/servidor. Quando um serviço roda (geralmente servidor HTTP), rodando em um certo IP e há o cliente que é o outro serviço que executa uma chamada para o servidor. Geralmente são utilizados o protocolo HTTP e o REst. A chamada pode ser síncrona ou assíncrona:

Síncrona: o cliente envia a sua requisição e o servidor envia a resposta. O cliente aguarda trancado esperando a resposta e só vai prosseguir a execução do programa quando a resposta for obtida.

Assíncrona: neste modelo você dispara e esquece. O cliente faz a requisição e isso roda em background. Quando a resposta chegar, ela será tratada.

Há um alto acoplamento entre cliente e o servidor, pois o cliente necessita conhecer exatamente o servidor e também há necessidade de transmitir os dados da forma esperada, tratando a resposta ou o erro. Ambos serviços precisam ser responsivos no momento. Uma falha no servidor gera uma falha no cliente.



02:34



07:19

PALAVRA-CHAVE

Pub/Sub: É um serviço de mensagens em tempo real totalmente gerenciado que permite o envio e o recebimento de mensagens entre aplicativos independentes.



07:28

“Eu não dependo mais de uma versão específica do meu servidor; eu não dependo de saber exatamente quem vai consumir essa mensagem.”



08:43

Fila de mensagens e tópico de mensagens

A fila de mensagens é uma estrutura em que cada mensagem produzida é entregue a um único consumidor, adequando-se para a distribuição de carga. Quando não há nenhum consumidor registrado ou disponível, a mensagem geralmente é armazenada e assim que um consumidor estiver disponível, a mensagem é entregue.



10:52

Os tópicos de mensagens são estruturas em que cada mensagem produzida é entregue a todos os consumidores registrados. Geralmente não há persistência das mensagens e somente consumidores registrados no momento em que a mensagem é gerada a recebem.

PALAVRA-CHAVE

Multicast: É uma técnica de transmissão de um pacote de dados para múltiplos destinos ao mesmo tempo. Durante uma transmissão, o transmissor envia os pacotes de dados somente uma vez, ficando a cargo dos receptores captarem esta transmissão e reproduzi-la.

Message broker

O message broker é um sistema especializado em recepção e envio de mensagens. Ele desconhece detalhes sobre os produtores e consumidores. O sistema é capaz de persistir nas mensagens e de entregar novamente uma mensagem em caso de falha do consumidor. Existem diversos serviços de mensageria bem conhecidos, como por exemplo: Kafka, ActiveMQ e RabbitMQ.

Um message broker confiável é essencial em uma arquitetura de microserviços, portanto, evite um ponto único de falha, ou seja, um componente que, caso falhe, impacta ou indisponibiliza o sistema todo. Também é importante lidar com problemas de escalabilidade, sendo necessário o monitoramento constante, já que tende a ser um componente do sistema bastante demandado.



11:26



16:03

Exemplos de código

Os exemplos de código a seguir utilizam o protocolo AMQP (Advanced Message Queuing Protocol) e são implementados pelo ActiveMQ, RabbitMQ e outros. São escritos em TypeScript, rodando sobre nodejs. Os exemplos apresentados estão disponíveis clicando [aqui](#).



16:32

PALAVRA-CHAVE

AMQP: É um protocolo de camada de aplicação padrão aberto para Message Oriented Middleware. As características definidoras do AMQP são: orientação, mensagem, roteamento, confiabilidade e segurança.



16:42



19:37

PALAVRA-CHAVE

Dotenv: A ferramenta é utilizada para orquestrar as variáveis ambiente de um projeto. O nome dela sugere o arquivo em que as informações ficarão, (dot) que é ponto em inglês acrescido de (env), então temos o arquivo (.env) que é composto de chaves e valores.



22:54

PALAVRA-CHAVE

ACK: Acknowledgement é um código de comunicação enviado por uma unidade receptora à uma estação transmissora, com o objetivo de confirmar que está pronta para receber um pacote de dados ou que o pacote enviado chegou sem erros.

Exemplo de saída

O professor inicia sua demonstração prática utilizando os códigos que foram disponibilizados anteriormente.



27:21

Tipos de garantia de entrega de mensagens

A primeira política de garantia de entregas é **no máximo uma vez**. Most-once delivery recebe uma mensagem e tenta entregar essa mensagem (no máximo uma vez) e, em caso de erro, ela é perdida. Nenhum estado é mantido, portanto é a implementação mais simples e rápida.

Já na entrega que ocorre **ao menos uma vez**, em caso de erro ou limite de tempo, ela será entregue novamente. Há necessidade de manter o estado no componente de entrega, podendo duplicar o processamento ou resultado, por isso é essencial que o tratamento de mensagens seja idempotente, isto é, não deixe o estado do sistema inconsistente se executado mais de uma vez.

Na política **exatamente uma vez**, há a garantia de que cada mensagem seja entregue uma única vez, mesmo que haja falhas ou limite de tempo. É o mecanismo mais complexo, pois exige estado em ambos os componentes de entrega e recepção. O componente de envio deve manter estado para retransmitir mensagens falhadas e o de recepção deve manter estado para ignorar mensagens que já tenham sido previamente enviadas.

36:09

34:22



PALAVRA-CHAVE

IOT: A Internet das Coisas descreve a rede de objetos físicos incorporados a sensores, software e outras tecnologias com o objetivo de conectar e trocar dados com outros dispositivos e sistemas pela internet.

EXERCÍCIO DE FIXAÇÃO

Sobre fila de mensagens, é correto afirmar que:

Exemplos de uso de mensagens

Nesta abordagem de notificar eventos de domínio, o sistema gera mensagens que representam eventos que ocorrem em um microsserviço para que outros microsserviços possam reagir de acordo. A atualização de caches de dados de outros serviços é semelhante a eventos de domínio, mas com os dados. É importante cuidar do versionamento/lock otimista. No processamento paralelo, os consumidores são “trabalhadores” e cada evento contém uma seleção do que será processado.

05:30



00:25



PALAVRA-CHAVE

Versionamento: É o processo de atribuir um nome único ou uma numeração única para indicar o estado de um programa de computador. Esses números são geralmente atribuídos de forma crescente e indicam o desenvolvimento de melhorias ou correção de falhas no software.

PALAVRA-CHAVE

Caixa de saída transacional: É um padrão que garante que os eventos são guardados num arquivo de dados antes de serem enviados para um mediador de mensagens. Se o objeto empresarial e os eventos correspondentes forem guardados na mesma transação de base de dados, é garantido que não serão perdidos dados. Tudo será consolidado ou tudo será revertido se existir um erro.



10:17

12:17



Serverless

As aplicações serverless necessitam de um servidor para rodar, porém elas não sabem qual servidor vai rodá-las. O conceito de serverless é geralmente relacionado ao FaaS (Function as a service), mas o conceito pode ser considerado mais amplo. Os serviços gerenciados (bases de dados, buscas, mensageria) também podem ser considerados serverless.

“ Geralmente, em produção, a gente tem um servidor de base de dados principal e a gente tem uma réplica. ”



14:15

Aplicações serverless retiram do operador do sistema a responsabilidade de gerenciar a infraestrutura do sistema (atualizações de segurança do sistema operacional, atualizações do software de base e administração de capacidade ou escala). Assim, o desenvolvedor/operador pode focar-se apenas na aplicação.

PALAVRA-CHAVE

Log4shell: É uma vulnerabilidade de dia zero no Log4j, uma estrutura de registro Java popular, envolvendo execução arbitrária de código.



17:52



20:54



Knative

Neste modelo, o desenvolvedor empacota funções, geralmente utilizando um container (como o Docker). Na medida que há demanda, o ambiente aloca recursos para executar a função. Quando a demanda cessa, o ambiente libera recursos. É adequado para funções de processamento, não para sistemas de persistência de dados. O FaaS tem como vantagens:

Otimização de custos: somente será cobrado quando houver demanda;

Escala flexível: a infraestrutura vai alojar mais recursos com o aumento de demanda, e desalocar recursos desnecessários, inclusive até chegar ao ponto de nenhum recurso alocado, ou seja, custo zero.

É uma plataforma com diversos componentes de construção baseados no Kubernetes. Ela apresenta um conjunto de elementos de middleware, os quais estão ligados ao desenvolvimento de aplicativos relacionados às tecnologias atuais. Uma das principais características da plataforma é que ela permite a atuação dentro de um contêiner, o que favorece a autenticação e a segurança da nuvem.

As desvantagens, são:

- Aumento na complexidade da infraestrutura;
- Difícil prever o custo final, pois depende da demanda;
- Maior dificuldade na depuração;
- O código de cada função é isolado em um container;
- Quando é necessário aumentar a escala, pode ocorrer um atraso devido ao tempo necessário para inicializar a função;
- O tempo de “aquecimento” da função pode impactar na experiência do usuário. O Java é um exemplo notável.

O Knative também:

- Expõe cada função em um containers (Docker);
- O Knative atribui uma URL para invocar cada função;
- A função deve levantar um servidor HTTP;
- O Knative roteia uma requisição para a função;
- Como a única responsabilidade da função é escutar em uma porta e retornar uma resposta, ela pode ser escrita em qualquer linguagem de programação.

Dockerfile - Parte I

O professor inicia a demonstração prática para ilustrar a funcionalidade do Docker.



PALAVRA-CHAVE

DNS: Convertem solicitações de nomes em endereços IP, controlando qual servidor um usuário final alcançará quando digitar um nome de domínio no navegador da web. Essas solicitações são chamadas consultas.

EXERCÍCIO DE FIXAÇÃO

O que é versionamento?

AULA 3 • PARTE 3

Dockerfile - Parte II

O professor dá continuidade à sua demonstração prática para ilustrar a funcionalidade do Docker.



00:25

01:19



“Eu poderia fazer a requisição ao meu serviço sem saber de antemão a URL.”

PALAVRA-CHAVE

Pod: É uma abstração do Kubernetes que representa um grupo de um ou mais contêineres de aplicativos (como Docker) e alguns recursos compartilhados para esses contêineres.



03:25

08:38



“Microsserviços é um assunto muito importante no desenvolvimento atual, porque é uma evolução natural.”

“Alguns segundos são suficientes para muitos usuários simplesmente desistirem daquele site.”



09:21

10:05



“Se você trabalha numa empresa que não abraça realmente a ideia de microserviços como uma solução, não vai funcionar.”

Exemplos de código - Parte I

O professor apresenta outro exemplo de código. Durante a demonstração prática, o professor ilustra diversos aspectos de microsserviços. O código apresentado está disponível clicando [aqui](#).



13:44

26:28



PALAVRA-CHAVE

PUT: Cria um novo recurso ou substitui uma representação do recurso de destino com os novos dados.

PALAVRA-CHAVE



33:43

Query: É um pedido de uma informação ou de um dado. Esse pedido também pode ser entendido como uma consulta, uma solicitação ou uma requisição.

EXERCÍCIO DE FIXAÇÃO

O que é FaaS?



37:27

PALAVRA-CHAVE



JSON: É um formato que armazena informações estruturadas e é principalmente usado para transferir dados entre um servidor e um cliente. O arquivo é basicamente uma alternativa simples e mais leve ao XML (Extensive Markup Language), que tem funções similares.

AULA 3 • PARTE 4

Exemplos de código - Parte II

O professor dá continuidade à sua demonstração prática, ilustrando como se dá a interação entre os microsserviços.



00:25

04:10



Toda vez que o serviço de produtos mudar de endereço, então o Consul vai me notificar e eu vou atualizar a URL que está rodando o meu serviço de produtos.



05:56

“ O nosso sistema aqui, ele é pequeno. Se ele crescesse muito, talvez tivesse que refatorar ele. **”**



19:02

PALAVRA-CHAVE

Postman: É uma ferramenta que dá suporte à documentação das requisições feitas pela API. Ele possui ambiente para a documentação, execução de testes de APIs e requisições em geral.

Resumo da disciplina

Veja, nesta página, um resumo dos principais conceitos vistos ao longo da disciplina.



Avaliação

Veja as instruções para realizar a avaliação da disciplina.

Já está disponível o teste online da disciplina. O prazo para realização é de **dois meses a partir da data de lançamento das aulas**.

Lembre-se que cada disciplina possui uma avaliação online.
A nota mínima para aprovação é 6.

Fique tranquilo! Caso você perca o prazo do teste online, ficará aberto o teste de recuperação, que pode ser realizado até o final do seu curso. A única diferença é que a nota máxima atribuída na recuperação é 8.

