Fundamentos de Arquitetura de Software

Arquitetura de Software -
Atividade Supervisionada 4

Prof. Gilmar Ferreira Arantes

Alunos: João Gabriel Cavalcante, Jair Souza Meira Rodrigues, Leonardo Moreira de Araújo, Matheus Franco Cascão, Vitor Paulo Eterno Godoi

2025

INF
INSTITUTO DE INFORMÁTICA

UFG
UNIVERSIDADE FEDERAL DE GOIÁS

Sumário

Fundamentos da Arquitetura de Software

Modularidade

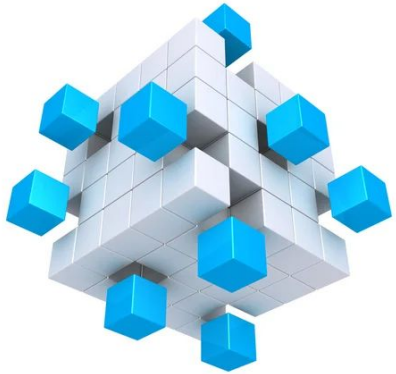
INF
INSTITUTO DE INFORMÁTICA

O que é?

- Separação do sistema em módulos **independentes** e **coesos**;
- Cada módulo é responsável por uma **função específica**.

Benefícios:

- Facilidade de **manutenção;**
- **Reutilização** de código;
- Melhor **legibilidade** e **organização;**
- Trabalho em equipe **facilitado;**
- **Evolução** e **escalabilidade** do sistema.



Modularidade

Características cruciais:

- **Alta coesão:** foco em uma única responsabilidade;
- **Baixo acoplamento:** pouca dependência de outros módulos;
- **Interfaces bem definidas:** comunicação clara e estável.

Aplicação:

- Utilizada em **Arquitetura em Camadas**, **Microservices** e **DDD;**
- Favorece **testabilidade** e **deploy** independente.

Desafios:

- Definir **limites** corretos dos módulos;
- Gerenciar **dependências** e **integração;**
- Evitar excesso de **fragmentação.**

Escalabilidade

INF
INSTITUTO DE INFORMÁTICA

O que é?

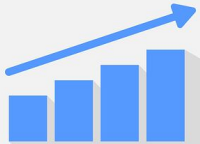
- Capacidade do sistema de **manter desempenho** à medida que cresce;
- Lida com o aumento de **usuários**, **dados** ou **requisições.**

Tipos:

- **Escalabilidade Vertical (Scale Up):**
  - Aumenta recursos de uma única máquina.
- **Escalabilidade Horizontal (Scale Out):**
  - Adicionar mais máquinas ao sistema.

Fatores que afetam a escalabilidade:

- Design da **arquitetura** e **banco de dados;**
- Uso eficiente de **recursos** e **balanceamento** de carga;
- **Desacoplamento** entre componentes.



Escalabilidade

Práticas benéficas:

- Uso de **microsserviços** e **filas assíncronas;**
- **Cache** eficiente (ex: Redis);
- Estratégias de **replicação** e **particionamento** de dados.

Escalabilidade e arquiteturas:

- Sistemas bem escaláveis geralmente usam:
  - **Arquiteturas distribuídas;**
  - **Serviços independentes;**
  - **Infraestrutura em nuvem** (auto scaling, containers).

Desafios:

- Complexidade de **comunicação** e **orquestração**;
- **Consistência** de **dados** em ambientes distribuídos;
- **Custos crescentes** com recursos e manutenção.

Manutenibilidade

O que é?

- Facilidade com que um sistema pode ser **modificado**, **corrigido** ou **adaptado;**
- Inclui correções de bugs, melhorias e novas funcionalidades.

Importância:

- **Reduz** o custo e tempo de **manutenção;**
- **Facilita evolução** contínua do sistema;
- **Ajuda** no **entendimento** do código por novos desenvolvedores.

Fatores influentes:

- **Qualidade do código** (clareza, simplicidade);
- **Modularidade** e **organização** do sistema;
- **Documentação** e **testes** adequados.

Manutenibilidade

INF
INSTITUTO DE INFORMÁTICA

Boas práticas:

- Código **limpo** e **padronizado;**
- Testes **automatizados** (unitários, integração);
- Arquitetura **desacoplada** e **bem definida;**
- Documentação **clara** e **atualizada.**

Benefícios:

- Alta manutenibilidade favorece:
  - **Reusabilidade;**
  - **Escalabilidade;**
  - **Segurança e confiabilidade.**

Desafios:

- Sistemas legados com **alto acoplamento;**
- Falta de **padrões** e **boas práticas** desde o início;
- Dificuldade em manter **documentação viva.**

Segurança

INF
INSTITUTO DE INFORMÁTICA

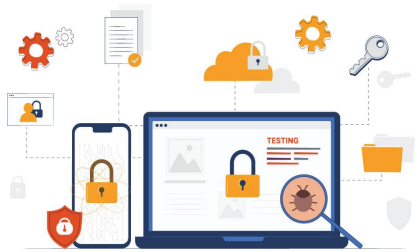
O que é?

Segurança na arquitetura de software se refere à incorporação de mecanismos e práticas que protejam o sistema contra ameaças externas e internas, garantindo:

- **Confidencialidade** (proteção contra acesso não autorizado);
- **Integridade** (garantia de que os dados não foram alterados indevidamente);
- **Autenticidade** (confirmação da identidade dos usuários e sistemas);
- **Auditoria** (capacidade de rastrear ações e eventos).

Pontos positivos

- **Protege ativos e dados sensíveis**, como informações de usuários e dados bancários;
- **Evita prejuízos financeiros e de reputação** decorrentes de falhas e ataques;
- **Conformidade com legislações**, como LGPD, GDPR, HIPAA, PCI-DSS;
- **Melhora a confiança dos usuários e parceiros** comerciais.



Segurança

Pontos negativos (ou desafios)

- **Adiciona camadas de complexidade à arquitetura** (ex: autenticação multifator, criptografia);
- **Impacta o desempenho** devido a operações adicionais (ex: criptografar dados, validações);
- **Manutenção contínua necessária** para lidar com novas vulnerabilidades e atualizações;
- **Pode gerar atrito na experiência do usuário** (ex: autenticações exigentes).

Boas práticas

- **APIs bem projetadas e documentadas** (ex: OpenAPI/Swagger);
- **Contratos de serviço estáveis e versionados**;
- **Middleware e gateways** que façam a mediação entre formatos e protocolos diferentes;
- **Uso de mensageria e filas** para desacoplar integrações (ex: Kafka, RabbitMQ);
- **Adoção de padrões abertos** (ex: REST, GraphQL, OAuth2).

Segurança

Ferramentas

- **Gerenciamento de Identidade e Acesso (IAM)**
  - **Auth0, Keycloak, Okta, Firebase Auth** → Implementam autenticação, autorização e Single Sign-On (SSO)
- **Criptografia e Gerenciamento de Segredos**
  - **Vault (HashiCorp)**: Armazena e protege tokens, chaves e segredos
  - **AWS KMS / Azure Key Vault / Google Secret Manager**: Criptografia gerenciada na nuvem
- **Análise de Vulnerabilidades e Segurança de Código**
  - **SonarQube, Snyk, OWASP Dependency-Check, Veracode** → Detectam falhas e vulnerabilidades em código
- **Firewall de Aplicação Web (WAF)**
  - **Cloudflare WAF, AWS WAF, ModSecurity** → Bloqueiam ataques como XSS e SQL Injection em tempo real
- **Auditoria e Monitoramento de Segurança**
  - **ELK Stack (Elasticsearch, Logstash, Kibana), Splunk, Datadog** → Observabilidade com foco em logs e eventos suspeitos

Desempenho

INF
INSTITUTO DE INFORMÁTICA

O que é?

Desempenho é a **capacidade do sistema de responder rapidamente**, com uso eficiente dos recursos computacionais (CPU, memória, I/O, rede), mesmo sob carga elevada. Métricas comuns:

- **Tempo de resposta;**
- **Throughput** (requisições processadas por segundo);
- **Latência;**
- **Uso de CPU/memória;**
- **Escalabilidade.**

Pontos positivos

- **Melhora a experiência do usuário final** (menos espera, respostas mais rápidas).
- **Reduz custos de infraestrutura**, com melhor aproveitamento de recursos.
- **Maior competitividade e satisfação do cliente.**
- **Preparação para crescimento** (número de usuários, volume de dados, requisições).

Desempenho

Pontos negativos (ou desafios)

- **Otimizações prematuras** podem tornar o código difícil de manter;
- **Trade-offs com outras qualidades**: segurança, legibilidade, manutenibilidade;
- **Testes de desempenho complexos e caros**;
- **Problemas de performance muitas vezes só aparecem em produção**.

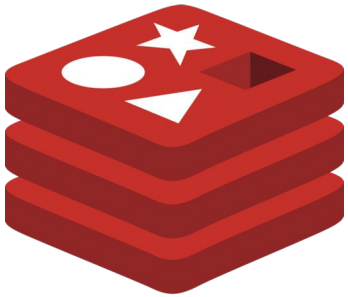
Boas práticas

- **Cache** de dados e respostas (ex: Redis, CDN, ETag).
- **Escalabilidade horizontal** com balanceamento de carga.
- **Monitoramento e observabilidade ativa** (ex: Prometheus, Grafana, APMs).
- **Técnicas de Lazy Loading e processamento assíncrono**.
- **Arquiteturas orientadas a eventos** para melhorar a eficiência de resposta.
- **Uso adequado de bancos de dados e índices**.

Desempenho

Ferramentas

- **Monitoramento e APM (Application Performance Monitoring)**
  - **New Relic, Datadog, Dynatrace, Prometheus + Grafana** → Medem latência, throughput, gargalos e erros
- **Cache**
  - **Redis, Memcached, Varnish** → Melhora o tempo de resposta e reduz carga em banco de dados
- **Balanceamento de Carga**
  - **NGINX, HAProxy, AWS Elastic Load Balancer** → Distribuem requisições de forma eficiente entre instâncias
- **Testes de Carga e Stress**
  - **Apache JMeter, Gatling, k6, Locust** → Simulam usuários e avaliam performance sob alta carga
- **CDN (Content Delivery Network)**
  - **Cloudflare, Akamai, Fastly, Amazon CloudFront** → Reduzem latência ao servir conteúdo mais próximo do usuário



Interoperabilidade

INF
INSTITUTO DE INFORMÁTICA

O que é?

Interoperabilidade é a **capacidade de sistemas distintos se comunicarem e trabalharem juntos**, mesmo que construídos com linguagens, plataformas ou arquiteturas diferentes. Abrange:

- **Integração entre sistemas internos e externos**
- **Troca de dados em formatos padronizados (JSON, XML)**
- **Adoção de APIs e protocolos universais (HTTP, REST, SOAP, gRPC)**

Pontos positivos

- **Facilita integrações com parceiros e terceiros**.
- **Permite evolução gradual**, sem precisar reescrever sistemas legados.
- **Promove a modularidade**, separação de responsabilidades e reuso.
- **Habilita ambientes heterogêneos** (cloud, on-premises, dispositivos móveis).

Interoperabilidade

Pontos negativos

- **Dependência de sistemas externos** pode introduzir fragilidade e instabilidade.
- **Problemas de compatibilidade** entre versões, padrões ou formatos de dados.
- **Latência e falhas de comunicação**.
- **Segurança adicional necessária** na troca de dados entre sistemas.

Boas práticas

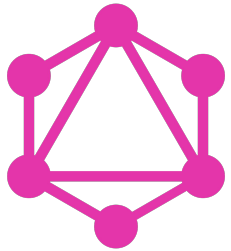
- **APIs bem projetadas e documentadas** (ex: OpenAPI/Swagger).
- **Contratos de serviço estáveis e versionados**.
- **Middleware e gateways** que façam a mediação entre formatos e protocolos diferentes.
- **Uso de mensageria e filas** para desacoplar integrações (ex: Kafka, RabbitMQ).
- **Adoção de padrões abertos** (ex: REST, GraphQL, OAuth2).

Interoperabilidade

Ferramentas

- **APIs e Contratos**
  - **Swagger / OpenAPI / Postman** → Design, documentação e testes de APIs RESTful
- **Mensageria e Integração Assíncrona**
  - **Apache Kafka, RabbitMQ, NATS, AWS SNS/SQS** → Permitem integração entre sistemas desacoplados
- **Protocolos e Padrões**
  - **gRPC (Google RPC), GraphQL, REST, SOAP (via WSDL)** → Facilitam comunicação entre serviços
- **Plataformas de Integração (iPaaS)**
  - **MuleSoft, Apache Camel, Dell Boomi, Zapier (para no-code)** → Facilitam integração entre múltiplos sistemas e serviços
- **Gateways e Middleware**
  - **Kong, Apigee, AWS API Gateway, NGINX Ingress Controller (K8s)** → Gerenciam APIs, autenticação, versionamento e roteamento



Resiliência

O que é?

Capacidade dos sistemas continuarem operando mesmo diante de falhas inesperadas.

Importância

- Garante alta disponibilidade, confiabilidade e boa experiência do usuário.
- Fundamental em áreas críticas: bancos digitais, nuvem, streaming.

Exemplos reais

- **Netflix**: Chaos Engineering (Chaos Monkey).
- **Nubank / Itaú**: Arquiteturas distribuídas para alta segurança.
- **AWS / Google Cloud**: Deploy gradual e balanceamento de carga.

Resiliência

Estratégias para implementar resiliência

Monitoramento Contínuo: Prometheus e Grafana para detectar problemas precocemente.

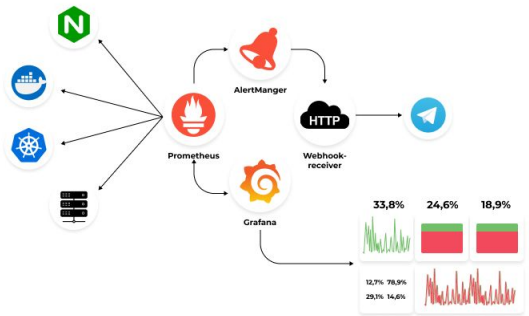
Retry & Circuit Breaker: Polly (.NET) / Hystrix (Java) para lidar com falhas transitórias.

Arquitetura Distribuída: Microsserviços e balanceamento de carga para isolar falhas.

Deploy Gradual: Blue-Green Deployment e Canary Releases.

Backup e Recuperação: Soluções como Veeam, AWS Backup para recuperação rápida.

Chaos Engineering: Simular falhas para validar a resiliência do sistema.



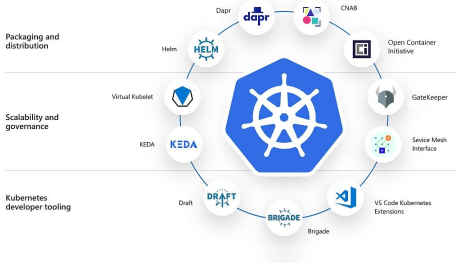
Resiliência

Principais ferramentas

- **Kubernetes:** Escalabilidade e auto-recuperação.
- **Prometheus & Grafana:** Monitoramento.
- **Chaos Monkey:** Teste de falhas em produção.

Boas práticas

- Avaliação contínua de vulnerabilidades (REOF).
- Testes de falha planejados e controlados.
- Implementar padrões como Retry, Timeout e Circuit Breaker.



Portabilidade

O que é?

Capacidade de rodar aplicações em diferentes ambientes ou plataformas sem grandes adaptações.

Importância

- Reduz custos de manutenção e desenvolvimento.
- Amplia a base de usuários e facilita integrações.

Exemplos reais

- **Google Docs / Office Online**: acesso via navegador em qualquer SO.
- **Flutter e React Native**: apps móveis para Android e iOS com o mesmo código.
- **Distribuições Linux (Ubuntu, Fedora)**: executam em diferentes hardwares.

Portabilidade

Estratégias para implementar resiliência

Uso de Linguagens Multiplataforma: Java, Python: compatíveis com diversos sistemas.

Separação de Código e Dependências: Evitar vincular o código a ambientes específicos.

APIs Padronizadas: Seguir padrões abertos para garantir compatibilidade.

Virtualização e Containers: Docker e Kubernetes: empacotam aplicações de forma independente.

Testes em Múltiplos Ambientes: Garantir funcionamento correto em diferentes sistemas.

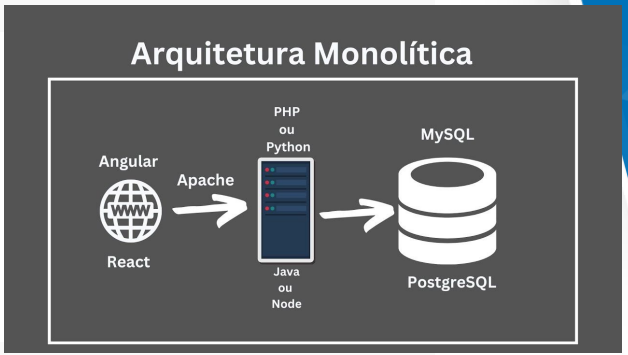
Design Responsivo: Aplicações web que se adaptam a diversos dispositivos.

Estilos Arquiteturais

Arquitetura Monolítica

O que é?

- Arquitetura monolítica é um modelo de desenvolvimento de software onde toda a aplicação é construída como um único bloco unificado, reunindo todas as suas funcionalidades (como interface, lógica de negócio e acesso a dados) em um único projeto executável.



Benefícios:

- **Arquitetura monolítica oferece simplicidade no desenvolvimento, teste e implantação, alta performance na comunicação interna, facilidade de gerenciamento centralizado e menor custo inicial de infraestrutura.**

Malefícios:

Arquitetura monolítica pode gerar dificuldades de manutenção e escalabilidade, implantações mais arriscadas, barreiras para adotar novas tecnologias e acoplamento excessivo entre módulos, tornando a evolução da aplicação mais lenta e complexa.

Arquitetura em Camadas

INF
INSTITUTO DE INFORMÁTICA

O que é?

- Arquitetura em camadas organiza a aplicação em níveis separados (como apresentação, negócio, persistência e banco de dados), cada um com responsabilidades específicas e comunicação entre eles de forma hierárquica.

Benefícios:

- **Arquitetura em camadas proporciona separação de responsabilidades, facilita a manutenção, melhora a organização do código, permite reuso de componentes e torna o sistema mais escalável e testável.**

Malefícios:

Arquitetura em camadas pode gerar sobrecarga de comunicação entre camadas, aumentar a complexidade do desenvolvimento, dificultar a performance em sistemas grandes e criar dependências rígidas entre as camadas.

Arquitetura Cliente-Servidor

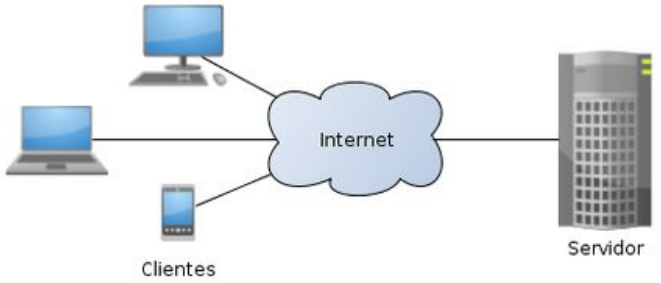
INF
INSTITUTO DE INFORMÁTICA

O que é?

- Arquitetura cliente-servidor divide o sistema em dois lados: o cliente, que faz requisições, e o servidor, que processa e responde, geralmente conectados via rede.

Benefícios:

- **Arquitetura cliente-servidor facilita a centralização de dados e serviços, permite o acesso remoto, simplifica a manutenção no servidor e possibilita escalabilidade através da adição de clientes.**



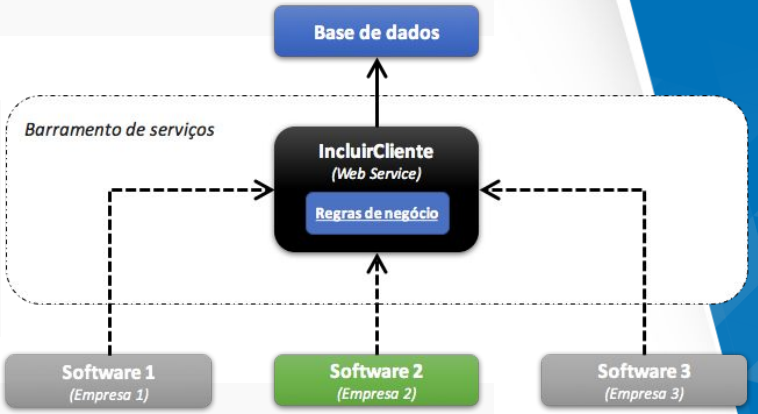
Malefícios:

Arquitetura cliente-servidor pode gerar sobrecarga no servidor, dependência de conexão de rede, possíveis gargalos de desempenho e maior complexidade na segurança dos dados transmitidos.

Arquitetura Orientada a Serviços - SOA

O que é?

- Arquitetura Orientada a Serviços (SOA) organiza a aplicação como um conjunto de serviços independentes que se comunicam entre si por interfaces bem definidas.



Benefícios:

- **Arquitetura orientada a serviços promove a reutilização de componentes, facilita a integração entre sistemas diferentes, aumenta a escalabilidade, e permite maior flexibilidade e manutenção modular.**

Malefícios:

Arquitetura orientada a serviços pode aumentar a complexidade de desenvolvimento, exigir maior esforço em orquestração e segurança, além de gerar latência na comunicação entre serviços.

Arquitetura de Microsserviços

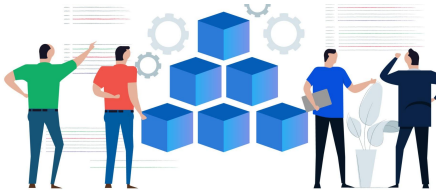
INF
INSTITUTO DE INFORMÁTICA

O que é?

- Um estilo de desenvolvimento em que a aplicação é estruturada como uma coleção de serviços pequenos e independentes.
- Cada serviço é responsável por uma funcionalidade específica do sistema, como por exemplo, a recomendação de produtos ou o processamento de pagamentos. Eles se comunicam entre si geralmente por meio de APIs ou mensageria.
- Podem ser programados em linguagens diferentes e usar bancos de dados diferentes se necessário.

Benefícios:

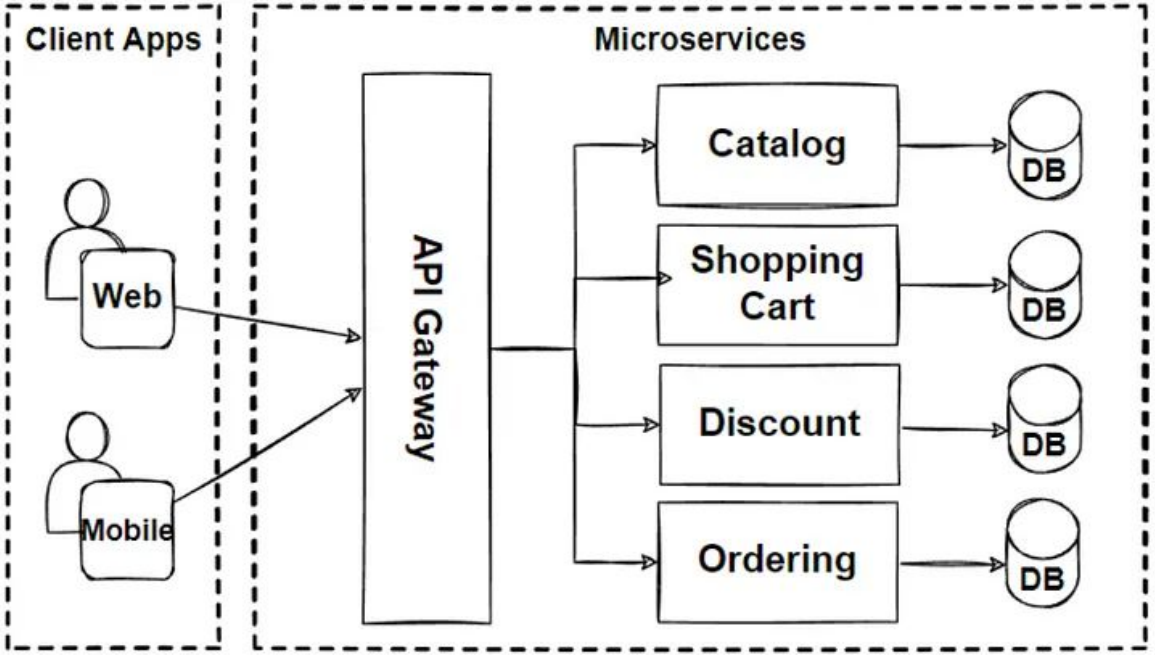
- **Maior disponibilidade, se um serviço falhar, os outros continuam;**
- **Facilidade na atualização de cada serviço de forma isolada**



Malefícios:

- **Complexidade na comunicação;**
- **Necessidade de gerenciamento eficiente;**
- **Monitoramento e logs centralizados se tornam críticos e fundamentais;**

Arquitetura de Microsserviços



- Diagrama representando a arquitetura de Microsserviços

Arquitetura de Microsserviços

Exemplo

A Uber tem milhares de microsserviços, e cada um é responsável por uma função pequena e específica, como:

- Serviço de matching (ligar motorista e passageiro),
- Serviço de mapas e geolocalização,
- Serviço de previsão de chegada,
- Serviço de pagamentos,
- Serviço de gestão de motoristas,
- Serviço de reputação (avaliação de usuários e motoristas).

"As Uber has grown to around 2,200 critical microservices, we experienced these tradeoffs first hand. Over the last two years, Uber has attempted to reduce microservice complexity while still maintaining the benefits of a microservice architecture. With this blog post we hope to introduce our generalized approach to microservice architectures, which we refer to as "Domain-Oriented Microservice Architecture" (DOMA)."

Fonte: https://www.uber.com/en-BR/blog/microservice-architecture/

Arquitetura Event-Driven

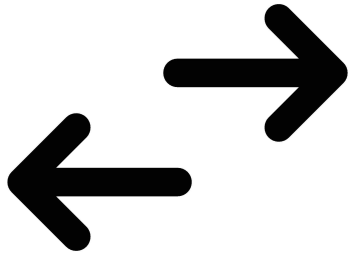
INF
INSTITUTO DE INFORMÁTICA

O que é?

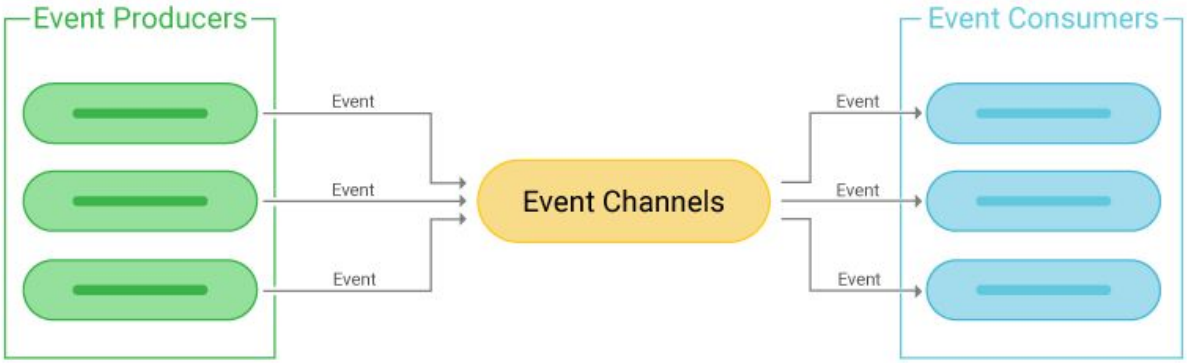
- Um estilo de desenvolvimento de sistemas onde **tudo gira em torno de eventos**, isto é, acontece algo a nível de usuário ou não e o sistema reage a isto.
- Nesta arquitetura, temos **produtores de eventos** (quem gera o evento) enviam notificações quando algo acontece. Enquantos, os **consumidores de eventos** (quem escuta o evento) recebem essas notificações e agem de acordo. O grande detalhe é que **o produtor não precisa saber quem vai consumir o evento. Essa desacoplagem deixa o sistema mais flexível e escalável.**

Exemplo

Evento	Reação
`PedidoCriado`	- Serviço de Estoque reserva os itens. - Serviço de Pagamento inicia a cobrança. - Serviço de Notificação envia e-mail de confirmação.
`PagamentoConfirmado`	- Serviço de Entregas separa o pedido. - Serviço de Notificação avisa o usuário.
`EntregaRealizada`	- Serviço de Avaliação envia um convite para o cliente avaliar a compra.



Arquitetura Event-Driven



- **Diagrama representando a arquitetura orientada a eventos.**

Arquitetura Event-Driven

Observação

- Os estilos de desenvolvimento podem ser combinados. Isso significa que podemos ter uma arquitetura baseada em microsserviços e event-driven simultaneamente. Por exemplo, imagine que em um sistema cada microsserviço é responsável por uma função específica e esses microsserviços comunicam entre si. No entanto, em vez deles chamarem uns aos outros diretamente por API's, eles podem se comunicar por eventos.

Benefícios:

- **Desacoplamento;**
- **Eficiência na resposta a eventos em tempo real;**

Malefícios:

- **Pode ser difícil de depurar e testar devido à natureza assíncrona;**
- **Complexidade de rastreamento;**

Arquitetura Serverless

O que é?

- Serverless é um jeito de construir aplicações sem gerenciar servidores, focando apenas na lógica do negócio, enquanto a nuvem lida com toda a infraestrutura.
- Basicamente, o código é feito em formas de função que são colocadas na nuvem. O provedor da nuvem executa essas funções sob demanda, escala automaticamente conforme o volume de acessos e o proprietário só paga pelo tempo que o código realmente foi executado, não paga pelo servidor ocioso.

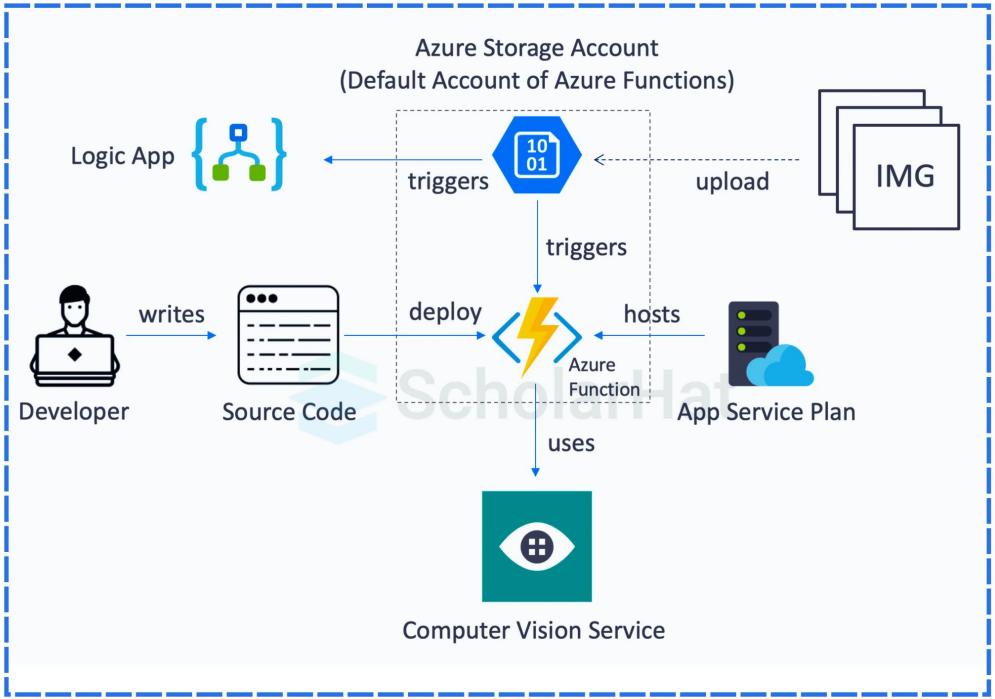
Benefícios:

- **Terceirização de responsabilidades como disponibilidade, escalabilidade, etc;**
- **Redução de custos, desenvolvimento mais rápido, sem overhead com configurações de servidor e etc;**

Malefícios:

- **Limitação de tempo, configuração e recurso;**
- **Cold start;**

Arquitetura Serverless



- **Diagrama representando a arquitetura serverless, utilizando Azure Functions.**

FIM!
OBRIGADO!!

INF
INSTITUTO DE INFORMÁTICA

UFG
UNIVERSIDADE FEDERAL DE GOIÁS