INSTITUTO DE INFORMÁTICA
Universidade Federal de Goiás

Fundamentos da Arquitetura de Software

Arquitetura de Software

Gilmar Ferreira Arantes

INSTITUTO DE INFORMÁTICA
UFG

UFG
UNIVERSIDADE FEDERAL DE GOIÁS

Grupo 9

- Giovanna Lyssa Rodrigues Borges Teles
- Gustavo Neves Piedade Louzada
- Halleffy Santos
- João Vitor da Costa Almeida
- Maria Eduarda de Campos Ramos

UFG

Agenda

1. Introdução
2. Fundamentos da Arquitetura de Software
   a. Modularidade
   b. Escalabilidade
   c. Manutenibilidade
   d. Segurança
   e. Desempenho
   f. Interoperabilidade
   g. Resiliência
   h. Portabilidade
3. Conclusão

UFG

Introdução

A **arquitetura de software** é a base estrutural que define como os sistemas são organizados, se comunicam e evoluem. Muito além de simples diagramas ou escolhas técnicas, ela representa **decisões estruturais críticas** que impactam diretamente a escalabilidade, a segurança, a interoperabilidade e a capacidade de evolução de uma solução. Compreender seus fundamentos é indispensável para projetar sistemas resilientes, eficientes e alinhados às demandas de negócios dinâmicos. Nesta apresentação, exploraremos os principais pilares da arquitetura de software, destacando como cada um contribui para construir sistemas que resistem ao tempo, às mudanças tecnológicas e aos desafios operacionais.

UFG

Fundamentos da Arquitetura de Software

Modularidade

- Modularidade é a divisão do software em partes, módulos independentes, cada um com sua interface, barramento e banco de dados

- Esse fundamento facilita a manutenção, reusabilidade e testes

- Princípios chaves: Independência, Alta Coesão, Baixo acoplamento, Gerenciamento de Dependências

UFG

Modularidade

- Como implementar:
  - Uso de “Princípios de Design”: SOLID
  - Estilos arquiteturais: Microsserviços, Arquitetura em camadas
  - Gerenciamento de dependências

- Ferramentas: Gerenciador de dependências (Maven, npm, pip), Microsserviços (Docker, Kubernetes), Controle de Versão (Git), CI/CD (Github Actions, Jenkins), Frameworks (Spring - Java, Angular/React - FrontEnd)

UFG

Escalabilidade

- A escalabilidade é a capacidade de um sistema lidar com o aumento da carga de trabalho, seja por meio da adição de recursos ou da adaptação da arquitetura
- Um sistema escalável consegue crescer de forma eficiente, sem perder desempenho, mesmo quando há aumento de usuários, dados ou requisições
- É um fundamento essencial para garantir que o sistema continue funcionando bem à medida que a demanda cresce, sem precisar ser totalmente refeito
- Pode envolver escalabilidade vertical (aumento de recursos em uma máquina) ou horizontal (adição de mais máquinas/servidores para dividir a carga)

UFG

Como garantir a escalabilidade?

- Planejar a arquitetura pensando em crescimento desde o início, evitando acoplamentos desnecessários que dificultem a expansão
- Utilizar bancos de dados distribuídos que suportam particionamento de dados (*sharding*) ou replicação
- Implementar cache em pontos estratégicos para reduzir a carga no banco de dados
- Adotar balanceadores de carga para distribuir requisições entre servidores
- Preferir arquiteturas baseadas em microsserviços, permitindo o crescimento independente de partes do sistema
- Monitorar constantemente o desempenho e utilizar estratégias de autoescalabilidade

UFG

Exemplos

- Alguns sistemas conhecidos que aplicam esse fundamento:
  - Amazon – A plataforma da Amazon escala automaticamente seus serviços conforme o volume de acesso, usando técnicas como *sharding* de banco de dados e replicação de servidores
  - Instagram – Utiliza cache pesado, banco de dados distribuído e microsserviços para conseguir escalar e atender a bilhões de interações diárias
  - Google Search – Distribui índices de pesquisa em milhares de servidores para atender bilhões de buscas diárias

UFG

Ferramentas

- Algumas ferramentas que podem auxiliar na **escalabilidade** de sistemas:
  - AWS Auto Scaling e Azure Scale Sets – Ajustam a quantidade de servidores automaticamente, de acordo com a demanda
  - Redis e Memcached – Soluções de cache que aliviam a pressão em bancos de dados e servidores
  - NGINX e HAProxy – Balanceadores de carga que distribuem o tráfego entre múltiplos servidores
  - Cassandra e MongoDB – Bancos de dados NoSQL que suportam particionamento de dados e replicação, permitindo escalabilidade horizontal

UFG

Manutenibilidade

- Manutenibilidade ou modificabilidade é a facilidade de entender um sistema e alterá-lo, corrigi-lo e incrementá-lo
- Garante um desenvolvimento e uma organização mais eficiente do sistema ao longo de todo o seu ciclo de vida
- Reduz custos de implementação e permite que mudanças sejam implementadas rapidamente
- Princípios chaves: Modularidade, Testabilidade, Legibilidade, Alta Coesão, Baixo Acoplamento e Documentação

UFG

Manutenibilidade

- Como implementar:
  - Uso de “Princípios de Design”: SOLID, DRY, KISS, Clean Code
  - Fundamentos de arquitetura: Modularidade, Testabilidade
  - Documentação: Requisitos, Arquitetura, Teste, Versionamento

- Ferramentas: CI/CD (Github Actions), Versionamento (Git), Teste Automatizado (JUnit, pytest)

UFG

Segurança

- É um conjunto de práticas, padrões e decisões tomadas desde as fases iniciais do desenvolvimento de um sistema para garantir que ele seja resistente a ameaças e vulnerabilidades

- Trata-se de um fundamento essencial para proteger dados, garantir a integridade do sistema e evitar ataques cibernéticos

UFG

Como garantir a segurança?

- Princípio do menor privilégio (conceder apenas as permissões estritamente necessárias)
- Validação de dados de entrada
- Autenticação forte (com o uso de *MFA* e senhas seguras)
- Gerenciamento seguro de sessão
- Criptografia de dados
- Testes de segurança
- Monitoramento contínuo
- Atualização de bibliotecas e dependências
- Treinamentos em segurança

UFG

Exemplos

- Alguns sistemas conhecidos que aplicam esse fundamento:
  - Sistemas bancários – Utilizam autenticação multifator, criptografia forte e gerenciamento seguro de sessão
  - Plataformas em nuvem (AWS, Azure, GCP) – Princípio do menor privilégio, controle granular de permissões, criptografia de dados
  - Sistemas governamentais – Controle de permissões, gerenciamento seguro de sessão, rastreabilidade, monitoramento contínuo

UFG

Ferramentas

- A seguir temos algumas ferramentas que podem auxiliar a garantir a **segurança** em sistemas:

    - Snyk – Detecta e corrige vulnerabilidades em dependências de projetos
    - Nmap – Auditoria de segurança de redes, muito útil para servidores e aplicações expostas
    - SonarQube – Análise contínua de código para encontrar vulnerabilidades e bugs
    - Burp Suite – Conjunto de ferramentas para testes de penetração em aplicações web
    - Aqua Security – Segurança para aplicações em nuvem e containers
    - Kubernetes e Docker – Permite a criação de ambientes isolados e seguros para aplicações

UFG

Desempenho

- Resposta rápida a solicitações
- Consumo de recursos de maneira eficiente
- O desempenho em projetos de software é essencial para garantir que sistemas sejam rápidos, eficientes e escaláveis

UFG

Como garantir um bom desempenho?

- Otimização de código
- Uso de cache
- Banco de dados próprio e otimizado para as necessidades
- Monitoramento contínuo
- Testes de desempenho
- Arquitetura eficiente

UFG

Exemplos

- Alguns sistemas conhecidos que aplicam esse fundamento:
  - Videogames – Funcionam em ambientes (consoles, computadores) com recursos limitados e que, portanto, precisam ter seus usos otimizados
  - Deepseek AI – Surgiu como forte concorrente ao ChatGPT e destacou-se por entregar resultados semelhantes a um custo muito menor de funcionamento
  - Serviços de streaming (Netflix, Prime Video, etc) – Dificilmente apresentam travamentos durante o streaming de vídeos por conta da aplicação de arquiteturas eficientes, uso de cache distribuído, entre outros

UFG

Ferramentas

- A seguir temos algumas ferramentas que podem auxiliar a garantir um **bom desempenho** em sistemas:

    - Prometheus + Grafana – Permite o monitoramento de métricas do sistema
    - Lighthouse (Google) – Avalia o desempenho de aplicações web (velocidade, boas práticas, etc)
    - Apache JMeter – Simula cargas pesadas para testar desempenho de servidores, aplicações e redes
    - New Relic – Monitora performance de aplicações em tempo real
    - Locust – Ferramenta de teste de carga baseada em Python

UFG

Interoperabilidade

- A arquitetura de um sistema deve permitir que ele se comunique com diferentes tecnologias e outros sistemas, de modo que haja uma integração fluida e compatibilidade entre eles

- É um fundamento essencial para que diferentes sistemas possam se comunicar e trocar informações

- É a capacidade de diferentes sistemas de software trabalharem juntos, conseguindo se comunicar, trocar dados e funcionar juntos de maneira eficaz, mesmo que tenham sido desenvolvidos separadamente, por organizações diferentes e/ou com tecnologias distintas

UFG

Como garantir a interoperabilidade?

- Utilizar APIs permite que sistemas troquem dados de forma padronizada, segura e eficiente
- Adotar formatos padronizados de dados, como JSON e XML, para a troca de dados, garantindo que diferentes sistemas possam interpretar e processar as informações corretamente
- Implementar protocolos de comunicação estabelecidos, como HTTP, REST e MQTT, facilita consideravelmente a troca de dados entre aplicações
- Uma arquitetura orientada a serviços permite que sistemas independentes se comuniquem por meio de serviços bem definidos
- Camadas de *middleware* podem auxiliar na conexão entre sistemas distintos
- Utilizar ontologias e padrões comuns para garantir que os sistemas compreendam corretamente o significado dos dados trocados

UFG

Exemplos

- Alguns sistemas conhecidos que aplicam esse fundamento:
    - Plataformas Governamentais – O iD Goiás é uma ferramenta de autenticação que permite acesso seguro a serviços digitais do governo, garantindo integração entre sistemas públicos
    - PIX – Sistema de pagamentos instantâneos que permite integração entre bancos e o Banco Central via interfaces abertas e seguras (APIs)
    - Serviços em Nuvem – Plataformas, como a AWS, oferecem interoperabilidade entre diferentes sistemas, permitindo que aplicações de diversas empresas se comuniquem de maneira segura e eficiente

UFG

Ferramentas

- A seguir temos algumas ferramentas que podem auxiliar a garantir a **interoperabilidade** entre sistemas:

    - Postman e Insomnia – Ajudam a criar e gerenciar APIs, facilitando a integração entre sistemas distintos
    - Swagger e OpenAPI – Facilitam o entendimento e consumo de APIs por meio de uma documentação padronizada
    - REST, HTTP, SOAP e GraphQL – Protocolos de comunicação que permitem troca de dados entre aplicações de forma padronizada
    - Apache Kafka e RabbitMQ – Apoiam a comunicação assíncrona entre sistemas
    - MongoDB, PostgreSQL e MySQL – Oferecem suporte a diferentes formatos de dados e integração com múltiplas aplicações
    - XML, JSON, YAML e CSV – Formatos de troca de dados que permitem uma comunicação padronizada entre sistemas

UFG

Resiliência

- O sistema deve continuar funcionando e oferecendo seus serviços aos usuários, mesmo diante de falhas, minimizando os impactos e recuperando-se rapidamente
- É sinônimo de ser tolerante a falhas
- É um dos fundamentos mais importantes para assegurar alta disponibilidade, confiabilidade e continuidade dos serviços, mesmo diante de falhas
- Trata-se da capacidade do sistema continuar operando corretamente, ou se recuperar rapidamente, mesmo após falhas, ataques, sobrecargas ou eventos inesperados

UFG

Como garantir a resiliência?

- Utilizar ferramentas de monitoramento contínuo pode ajudar na detecção de problemas antes que eles afetem o usuário final, além de permitir o monitoramento de logs, eventos e métricas em tempo real
- Implementar um design voltado para falhas, de forma que o sistema falhe de maneira controlada, sem “quebrar” completamente (assumindo que falhas inevitavelmente ocorrerão)
- Adotar uma arquitetura distribuída, como microsserviços, garante que falhas em um componente não afetem todo o sistema
- Ter componentes duplicados (redundância) ou distribuídos para garantir disponibilidade – por exemplo, múltiplas instâncias de servidores e bancos de dados replicados
- Utilizar balanceadores de carga para distribuir chamadas entre servidores/instâncias (se um nó cair, outro assume automaticamente)

UFG

Como garantir a resiliência?

- Repetir requisições automaticamente quando houver falhas transitórias e adicionar *exponential backoff* para evitar sobrecarga dos serviços
- Implementar padrões de *Circuit Breaker*, como Polly e Hystrix, para interromper automaticamente chamadas a serviços que estão falhando e evitar que todo o sistema seja afetado com uma parte instável
- Ajustar automaticamente a quantidade de recursos com base na demanda, evitando quedas por sobrecarga (*auto-scaling*)
- Evitar dependências diretas entre partes do sistema, adotando o desacoplamento de componentes
- Realizar testes de resiliência por meio de *Chaos Engineering*, isto é, simular falhas em produção para testar a capacidade do sistema de se recuperar e continuar funcionando
- Implementar estratégias de backup automático e recuperação rápida para minimizar os impactos de falhas críticas

UFG

Exemplos

- Alguns sistemas conhecidos que aplicam esse fundamento:
  - Netflix – Implementa *Chaos Engineering* para simular falhas em ambiente de produção e testar a resiliência do sistema, além de adotar replicação geográfica para garantir alta disponibilidade e baixa latência
  - WhatsApp – Foi construído para funcionar bem mesmo em conexões instáveis ou de baixa qualidade, graças à sua arquitetura distribuída, replicação de dados e compressão eficiente
  - Mercado Livre – Adota práticas como autoescalabilidade, microsserviços, uso de filas (Kafka) e *fallback pages* para lidar com instabilidades, sendo resiliente a picos de acesso, como os que ocorrem na Black Friday, por exemplo

UFG

Ferramentas

- A seguir temos algumas ferramentas que podem auxiliar a garantir a **resiliência** em softwares:
  - Chaos Monkey – Ferramenta desenvolvida pela Netflix que simula falhas em ambiente de produção para testar a capacidade de recuperação e a resiliência de sistemas
  - Prometheus, Grafana e Datadog – Ferramentas de monitoramento que auxiliam a detectar falhas antes que elas ocorram, por meio da análise de métricas, logs e eventos em tempo real
  - Kubernetes – Plataforma de orquestração de containers que permite escalabilidade automática e recuperação de serviços em caso de falhas
  - NGINX e HAProxy – Balanceadores de carga que distribuem solicitações entre servidores para evitar sobrecarga em um único ponto e garantir alta disponibilidade
  - Hystrix – Biblioteca que implementa o padrão *Circuit Breaker*, evitando sobrecarga em serviços ao interromper chamadas para componentes que estão falhando
  - AWS – Plataforma de computação em nuvem que oferece infraestrutura distribuída em múltiplas zonas de disponibilidade, possibilitando recuperação rápida de dados e continuidade dos serviços, mesmo diante de falhas críticas

UFG

Portabilidade

- A portabilidade é a capacidade de um sistema ou aplicação ser transferido e executado em diferentes ambientes, plataformas ou infraestruturas, com o mínimo de adaptação possível
- Um sistema portável facilita a migração entre servidores físicos, máquinas virtuais, provedores de nuvem ou diferentes sistemas operacionais
- É um fundamento importante para garantir flexibilidade, redução de dependências específicas e facilitar a evolução da infraestrutura no futuro
- A portabilidade evita o chamado *vendor lock-in*, onde a aplicação fica presa a uma tecnologia ou fornecedor específico

UFG

Como garantir a portabilidade?

- Adotar linguagens de programação e tecnologias que são suportadas em diferentes ambientes
- Evitar usar recursos proprietários de uma plataforma específica que não possam ser facilmente replicados em outros lugares
- Utilizar containers, como Docker, para empacotar aplicações de forma padronizada e portátil entre diferentes sistemas
- Utilizar ferramentas de automação de infraestrutura (como Terraform e Ansible) para configurar ambientes de forma reprodutível e independente da plataforma
- Escrever código com atenção às diferenças entre sistemas operacionais, especialmente para arquivos, redes e permissões

UFG

Exemplos

- Alguns sistemas conhecidos que aplicam esse fundamento:
  - Spotify – Usa containers Docker e automação para migrar entre diferentes provedores de nuvem com agilidade
  - WordPress – Plataforma de gestão de conteúdo que roda facilmente em Windows, Linux, servidores locais e nuvens públicas, como AWS e Google Cloud
  - Minecraft – Jogo projetado para rodar em diferentes sistemas operacionais (Windows, Linux, MacOS), mantendo comportamento consistente

UFG

Ferramentas

- Algumas ferramentas que podem auxiliar na **portabilidade** de sistemas:
    - Docker – Criação de containers que garantem a execução da aplicação em qualquer lugar
    - Podman – Alternativa ao Docker, compatível com OCI (Open Container Initiative), usado para containers portáteis
    - Terraform – Gerencia infraestrutura como código, facilitando a movimentação entre provedores de nuvem
    - Ansible – Automatiza configurações de servidores em múltiplos ambientes
    - Vagrant – Criação de ambientes de desenvolvimento portáveis, replicáveis e fáceis de mover
    - GitHub Actions – Automatiza testes e deploys para múltiplos ambientes e sistemas

UFG

Conclusão

Ao dominar os fundamentos da arquitetura de software, desenvolvedores e arquitetos **ampliam sua capacidade de criar soluções não apenas funcionais, mas sustentáveis e estratégicas**. A atenção à modularidade, escalabilidade, manutenibilidade, segurança, interoperabilidade, resiliência e portabilidade permite que sistemas evoluam com robustez e flexibilidade, antecipando falhas e abraçando a inovação.

UFG

Obrigado

Dúvidas ou sugestões?