Colégio Politec
Curso técnico em informática

TIME 02 – OWLOCK

Arquitetura de microsserviços e computação serveless

Lavínia, Guilherme e João Henrique

Americana-SP
2026

Arquitetura de Microsserviços

Antes de entender o que é uma arquitetura de microsserviços, é importante compreender o conceito de **arquitetura de software**. De forma simples, arquitetura é a maneira como um sistema é planejado e organizado para funcionar corretamente. Podemos fazer uma comparação com a construção de uma casa. Antes que ela seja construída, existe um projeto que define onde ficarão os quartos, a cozinha, o banheiro, as portas e as janelas. Cada ambiente possui uma função específica, mas todos trabalham juntos para formar uma única casa. Da mesma forma, a arquitetura de software define como um sistema será estruturado, como seus componentes serão organizados e como eles irão se comunicar para oferecer um funcionamento eficiente.

A **arquitetura de microsserviços** é um modelo de arquitetura de software em que uma aplicação é dividida em vários serviços menores e independentes, chamados de microsserviços. Em vez de concentrar todas as funcionalidades em um único programa, cada microsserviço é responsável por executar apenas uma função específica. Por exemplo, em um sistema de compras online, pode existir um microsserviço responsável pelo login dos usuários, outro pelo cadastro de clientes, outro pelo gerenciamento dos produtos, outro pelos pagamentos e outro pelo controle do estoque. Embora sejam independentes, todos esses serviços se comunicam por meio de APIs, permitindo que a aplicação funcione como um único sistema.

Podemos utilizar novamente o exemplo da casa para entender essa organização. Imagine que seja necessário reformar apenas a cozinha. Essa reforma não impede que as pessoas continuem utilizando o quarto, a sala ou o banheiro. Da mesma forma, em uma arquitetura de microsserviços, quando um serviço precisa ser atualizado ou corrigido, normalmente não é necessário interromper o funcionamento dos demais. Essa independência facilita a manutenção, reduz o impacto de falhas e permite que novas funcionalidades sejam adicionadas com mais rapidez.

Além da facilidade de manutenção, outra grande vantagem dos microsserviços é a escalabilidade. Se apenas uma parte do sistema estiver recebendo muitos acessos, como o serviço de pagamentos durante uma grande promoção, somente esse microsserviço pode receber mais recursos de processamento, sem que seja necessário ampliar toda a aplicação. Isso torna o uso dos recursos mais eficiente e reduz custos. Outro benefício é que diferentes equipes podem desenvolver e atualizar serviços distintos ao mesmo tempo, tornando o desenvolvimento mais ágil.

Apesar das vantagens, essa arquitetura também apresenta desafios. Como a aplicação é formada por diversos serviços independentes, é necessário garantir que todos consigam se comunicar corretamente, monitorar seu funcionamento e administrar uma infraestrutura mais complexa. Mesmo assim, devido à sua flexibilidade, facilidade de expansão e maior resistência a falhas, a arquitetura de microsserviços é amplamente utilizada em aplicações modernas executadas em ambientes de computação em nuvem, como os serviços oferecidos pelo Google Cloud.

Vantagens dos microsserviços

Os benefícios da integração da arquitetura de microsserviços são abrangentes. As principais vantagens incluem as mais notáveis:

- Escalabilidade aprimorada
- Maior agilidade e menor tempo de lançamento no mercado
- Isolamento de falhas e resiliência do sistema
- Eficácia em custo
- Integração de DevOps e entrega contínua
- Flexibilidade e inovação tecnológica

Escalabilidade aprimorada

As organizações dimensionam exatamente o que precisam com os microsserviços. Por exemplo, durante as épocas de pico, a empresa de plataforma de acomodações de viagem Airbnb expande os serviços de busca e reserva, mantendo outros serviços, como mensagens de hosts e sistemas de avaliações em capacidade normal.

Diferentes serviços usam estratégias de escala distintas com base em seus requisitos. Essa abordagem modular reduz os custos de infraestrutura e melhora a eficiência dos recursos em comparação com o dimensionamento de aplicações monolíticas inteiras.

Maior agilidade e tempo de lançamento no mercado mais rápido

Equipes pequenas e autônomas possuem serviços específicos de ponta a ponta. Cada equipe escolhe as melhores tecnologias para seu serviço e migra em seu próprio ritmo, sem esperar pela coordenação em toda a organização ou por ciclos de reimplantação.

As empresas podem testar ideias rapidamente, iterar rapidamente e responder às mudanças do mercado mais rapidamente,

desenvolvendo funcionalidades como novos serviços que se conectam aos existentes.

Isolamento de falhas e resiliência do sistema

Se um mecanismo de recomendação falhar, os usuários ainda podem procurar produtos, adicionar itens ao carrinho e finalizar a compra. Disjuntores e padrões semelhantes permitem que os serviços lidem com falhas sem problemas quando as dependências não respondem.

Essa resiliência é crucial para empresas onde os custos de downtime chegam a milhares de dólares por minuto. Raramente é possível que a aplicação inteira fique inativa, mesmo quando componentes individuais falham, porque as falhas permanecem isoladas.

Eficácia de custo

As organizações economizam dinheiro por meio do gerenciamento eficiente de recursos, dimensionando apenas os serviços que exigem maior capacidade em vez de aplicações inteiras. As equipes de desenvolvimento podem otimizar os custos selecionando o stack de tecnologia mais adequado para os requisitos específicos de cada serviço, evitando os custos de engenharia excessiva ou a aplicação uniforme de soluções de nível empresarial. O modelo pré-pago de microsserviços baseados em nuvem alinha os custos diretamente com o uso real.

Uma pesquisa da IBM, Microservices in the Enterprise, 2021, revelou que 87% de mais de 1.200 executivos de TI, executivos de desenvolvedores e desenvolvedores concordam que a adoção dos microsserviços vale o custo e o esforço.

Integração de DevOps e entrega contínua

As equipes de DevOps podem introduzir novos componentes sem dificuldades e sem causar downtime, graças à operação independente de cada serviço.

Testes automatizados, pipelines de implementação e infraestrutura como código (IaC) funcionam bem com microsserviços, criando uma cultura de lançamentos rápidos e confiáveis.

Flexibilidade e inovação tecnológica

Cada microsserviço pode usar tecnologias diferentes, incluindo linguagens de programação (por exemplo, Java, Python), frameworks e bancos de dados mais adequados para sua função específica. As equipes não ficam presas a uma única stack de tecnologia para toda a aplicação.

As organizações podem adotar novas tecnologias incrementalmente, experimentar ferramentas emergentes e aproveitar soluções especializadas onde elas fornecem mais valor.

Desvantagens dos microsserviços

- Embora ofereçam benefícios substanciais, há algumas desvantagens dos microsserviços que as organizações devem abordar:
- Complexidade dos testes
- Considerações de segurança
- Coordenação de comunicação
- Gerenciamento de dados

Complexidade dos testes

Testar uma aplicação baseada em microsserviços requer uma abordagem diferente em comparação com aplicações monolíticas. Todas as dependências, armazenamento em cache e acesso a dados precisam de verificação para um desempenho adequado. Os ambientes de teste envolvem vários serviços em execução simultaneamente com configuração e dados adequados.

À medida que os serviços crescem, os cenários de teste se expandem. Manter a consistência e a integridade dos dados de teste entre os serviços requer planejamento estratégico.

Considerações de segurança

A comunicação de rede entre microsserviços cria requisitos de segurança aprimorados. Os API Gateways precisam de autenticação e criptografia adequadas.

O gerenciamento de credenciais e tokens em vários microsserviços exige uma coordenação cuidadosa, e as equipes de segurança precisam de ferramentas de visibilidade robustas para monitorar a atividade em toda a arquitetura distribuída.

Coordenação de comunicação

As considerações de design tornam-se essenciais ao coordenar a comunicação entre os serviços. As chamadas entre serviços adicionam latência de rede em comparação com as chamadas de função em processo e esses atrasos se acumulam à medida que as solicitações fluem por vários serviços.

Funcionalidades de resiliência de rede, como tentar novamente, timeouts e tratamento de falhas, tornam-se práticas padrão. O gerenciamento de versões de API também exige um planejamento cuidadoso, pois os serviços evoluem de forma independente.

Gerenciamento de dados

Cada microsserviço que gerencia seu próprio banco de dados cria desafios de gerenciamento de dados. Manter a consistência quando as transações abrangem vários bancos de dados exige padrões específicos e relatórios precisam de abordagens diferentes quando os dados são distribuídos em vez de centralizados.

Computação Serveriess (Sem Servidor)

A **computação serverless** é um modelo de computação em nuvem que permite aos desenvolvedores criar, implantar e executar aplicações sem a necessidade de gerenciar servidores ou toda a infraestrutura de back-end. Apesar do nome *serverless* significar "sem servidor", isso não quer dizer que os servidores não existam. Na realidade, eles continuam sendo utilizados, porém toda a infraestrutura é administrada pelo provedor de serviços em nuvem (Cloud Service Provider – CSP), como AWS, Microsoft Azure ou Google Cloud. Dessa forma, o desenvolvedor não precisa configurar servidores, instalar sistemas operacionais, aplicar atualizações, monitorar recursos ou realizar manutenção, podendo concentrar seus esforços apenas no desenvolvimento da aplicação e na lógica de negócio.

Nesse modelo, o desenvolvedor escreve o código e o implanta na plataforma de nuvem. Sempre que ocorre uma solicitação, como um usuário acessando um site, enviando um formulário ou realizando uma compra, o provedor executa automaticamente o código

utilizando apenas os recursos necessários. Caso a demanda aumente, a infraestrutura é expandida automaticamente para suportar o maior número de requisições. Quando a demanda diminui, os recursos são reduzidos ou até mesmo desligados automaticamente, processo conhecido como **escala para zero (scale to zero)**.

Outra característica importante da computação serverless é seu modelo de cobrança. Diferentemente da infraestrutura tradicional, na qual um servidor permanece ligado continuamente, no serverless o usuário paga apenas pelo tempo em que o código permanece em execução e pelos recursos efetivamente consumidos. Dessa forma, elimina-se o custo de capacidade ociosa, tornando esse modelo mais econômico para aplicações com cargas de trabalho variáveis.

Como exemplo, considere uma aplicação que envia um e-mail sempre que um cliente realiza uma compra. Em um ambiente tradicional, seria necessário manter um servidor funcionando durante 24 horas por dia, mesmo que nenhuma compra fosse realizada. No modelo serverless, o código responsável pelo envio do e-mail é executado somente quando a compra acontece. Após concluir a tarefa, a execução é encerrada e os recursos são liberados automaticamente, gerando custos apenas pelos segundos em que a função esteve ativa.

O funcionamento da computação serverless depende principalmente do provedor de serviços em nuvem, que é responsável por disponibilizar servidores, armazenamento, bancos de dados e demais recursos necessários para a execução das aplicações. Além disso, cabe ao provedor realizar tarefas como atualizações do sistema operacional, aplicação de patches de segurança, monitoramento, planejamento de capacidade e manutenção da infraestrutura. Isso reduz significativamente a carga operacional das equipes de TI e permite que os desenvolvedores foquem exclusivamente na criação de novas funcionalidades.

A arquitetura serverless é baseada em **eventos**, sendo classificada como uma **Arquitetura Orientada a Eventos (Event-Driven Architecture – EDA)**. Nesse modelo, uma função é executada apenas quando ocorre um evento específico, como uma requisição HTTP, uma alteração em um banco de dados, o envio de uma mensagem ou o upload de um arquivo. Como consequência, os recursos computacionais são utilizados somente quando realmente necessários, aumentando a eficiência do ambiente.

Outra característica importante é que as aplicações serverless normalmente são **sem estado (stateless)**. Isso significa que cada execução da aplicação ocorre de forma independente, sem armazenar informações da execução anterior. Essa característica favorece a **programação assíncrona**, permitindo que diversas tarefas sejam executadas simultaneamente sem depender umas das outras, aumentando o desempenho e reduzindo o tempo de resposta das aplicações.

A comunicação entre os diferentes serviços normalmente é realizada por meio de **APIs RESTful**, que permitem a troca de informações utilizando o protocolo HTTP. Essa abordagem facilita a integração entre aplicações, bancos de dados e outros serviços disponibilizados pelos provedores de nuvem.

Além disso, o modelo serverless integra-se facilmente às práticas de **DevOps**, principalmente por meio de processos de **Integração Contínua (CI)** e **Entrega ou Implantação Contínua (CD)**. Como a infraestrutura é gerenciada automaticamente pelo provedor, os desenvolvedores podem automatizar grande parte do processo de desenvolvimento e implantação, reduzindo o tempo necessário para disponibilizar novas versões da aplicação.

Outro benefício importante é o **redimensionamento automático (Auto Scaling)**. Sempre que o número de usuários aumenta, a plataforma disponibiliza mais recursos automaticamente. Da mesma forma, quando a demanda diminui, os recursos são reduzidos, garantindo eficiência no uso da infraestrutura e melhor aproveitamento dos custos.

Algumas plataformas serverless também oferecem mecanismos de **autocura (Self-Healing),** capazes de detectar falhas automaticamente, reiniciar serviços ou executar novas instâncias da aplicação quando ocorre algum problema. Esse recurso aumenta a disponibilidade e a confiabilidade das aplicações.

Dentro do ecossistema serverless existem alguns modelos importantes. O **FaaS (Function as a Service)** é considerado um subconjunto do serverless e permite que os desenvolvedores executem pequenas funções independentes em resposta a eventos específicos, sem se preocupar com a infraestrutura. Embora os termos FaaS e serverless sejam frequentemente utilizados como

sinônimos, o FaaS representa apenas um dos serviços oferecidos pelo modelo serverless.

Outro modelo é o **BaaS (Backend as a Service)**, que disponibiliza funcionalidades de back-end prontas para uso, como autenticação de usuários, armazenamento em nuvem, gerenciamento de banco de dados e notificações push. Assim, o desenvolvedor pode concentrar seus esforços na construção da interface da aplicação, reutilizando serviços já disponibilizados pelo provedor.

Já o **SaaS (Software as a Service)** representa um modelo diferente. Nesse caso, o usuário utiliza um software pronto, disponibilizado pela internet pelo fornecedor do serviço, sem necessidade de desenvolver ou gerenciar a aplicação. Portanto, enquanto o serverless fornece uma plataforma para desenvolvimento e execução de aplicações, o SaaS entrega aplicações completas diretamente aos usuários finais.

Como toda a infraestrutura é administrada pelo provedor de nuvem, torna-se importante utilizar ferramentas de **monitoramento serverless**. Essas ferramentas permitem acompanhar a execução das funções, identificar falhas, medir desempenho, monitorar consumo de recursos e analisar métricas da aplicação. Embora o modelo reduza a responsabilidade do desenvolvedor sobre a infraestrutura, ele também reduz a visibilidade sobre o ambiente interno, tornando o monitoramento um componente essencial para garantir disponibilidade, desempenho e confiabilidade das aplicações.

Em resumo, a computação serverless representa uma evolução no desenvolvimento de aplicações em nuvem ao transferir o gerenciamento da infraestrutura para o provedor de serviços. Esse modelo oferece escalabilidade automática, cobrança baseada no consumo, redução dos custos operacionais, maior produtividade dos desenvolvedores e integração com práticas modernas de desenvolvimento, tornando-se uma das principais arquiteturas utilizadas em aplicações modernas baseadas em nuvem.

Vantagens

A computação serverless oferece aos desenvolvedores individuais e às equipes de desenvolvimento empresarial muitos benefícios técnicos e comerciais:

- **Maior produtividade do desenvolvedor:** como observado, o serverless possibilita que as equipes de desenvolvimento se

concentrem em escrever código, não em gerenciar a infraestrutura. Isso dá aos desenvolvedores mais tempo para inovar e otimizar suas funções de aplicativos front-end e lógica de negócios.

- **Pague somente pela execução:** o medidor é acionado quando a solicitação é feita e é encerrado quando a execução termina. Compare isso com o modelo de computação IaaS, em que os clientes pagam pelos servidores físicos, VMs e outros recursos necessários para executar aplicativos, desde o provisionamento desses recursos até o desatribuição explícita.
- **Desenvolva em qualquer linguagem:** o serverless é um ambiente multilinguagem que possibilita aos desenvolvedores programar em qualquer linguagem ou estrutura, seja Java, Python, JavaScript, node.js, em que estejam familiarizados.
- **Ciclos simplificados no desenvolvimento e na DevOps.** O serverless simplifica a implementação e, de modo mais amplo, o DevOps, pois os desenvolvedores não perdem tempo definindo a infraestrutura necessária para integrar, testar, entregar e implementar as construções de códigos na produção.
- **Desempenho econômico.** Em certas cargas de trabalho, como processamento paralelizáveis, processamento de fluxo e algumas tarefas de processamento de dados, a computação serverless pode ser mais rápida e mais econômica do que outras formas de computação.
- **Reduzir latência:** em um ambiente serverless, o código pode ser executado mais perto do usuário final, diminuindo a latência.
- **Visibilidade do uso:** as plataformas serverless dão visibilidade quase total dos tempos do sistema e do usuário e sabem agregar informações de uso de forma sistemática.

Desafios da computação serverless

Apesar dos benefícios óbvios de eficiência de custo e desenvolvimento acelerado, a computação serverless vem com um conjunto de desafios. Algumas desvantagens da computação serverless:

- **Dependência do fornecedor.** A natureza do serverless implica se comprometer com um único provedor de serviços em nuvem para implantação do código. Como resultado, os desenvolvedores são forçados a seguir o modelo oferecido pelo

fornecedor. O provedor determina o uso dos recursos, como limites de simultaneidade. Por esses motivos, a dependência do fornecedor pode significar falta de flexibilidade.

- **Monitoramento e depuração da perspectiva do código.** Como o provedor de serviços em nuvem gerencia a infraestrutura, você tem pouca ou nenhuma visibilidade do backend das suas operações. Consequentemente, é muito difícil monitorar um ambiente serverless sem uma ferramenta de monitoramento serverless dedicada. Além disso, com a arquitetura orientada por eventos de um ambiente serverless, identificar, recriar e corrigir bugs pode ser um desafio.
- **Latência.** Devido à natureza sem estado das aplicações em um ambiente serverless, pode ocorrer latência quando uma função é invocada pela primeira vez ou após um longo período de inatividade. A latência também pode ocorrer devido a um tempo limite ou quando muitas funções estão sendo executadas simultaneamente. Nesse caso, o provedor pode eliminar uma das funções, levando a uma falha. No lado do usuário, isso causa latência.
- **Customização e controle limitados.** Como os provedores serverless gerenciam a infraestrutura subjacente, pode haver limitações quanto à customização e ao controle do ambiente, como versões de tempo de execução disponíveis, alocação de memória e limites de tempo de execução.
- **Preocupações com a segurança.** Embora a computação serverless reduza a superfície de ataque, ela pode introduzir novos riscos à segurança. Os desenvolvedores precisam estar cientes dos riscos associados a serviços de terceiros, permissões de nível de função e vulnerabilidades no código da aplicação.
- **Condição sem estado.** As funções serverless são sem estado, o que significa que não retêm nenhum dado entre as invocações. Isso pode dificultar o gerenciamento do estado da aplicação, exigindo que os desenvolvedores dependam de armazenamento externo ou bancos de dados para manter o estado.
- **Previsibilidade de custo.** Embora a computação serverless possa ser econômica, pode ser difícil prever os custos, pois eles dependem de fatores como número de invocações, memória e tempo de execução. Picos inesperados de uso podem levar ao aumento dos custos.

- **Integração com sistemas existentes.** A integração de funções serverless com sistemas e arquiteturas existentes pode ser um desafio, principalmente ao lidar com aplicações legadas ou sistemas complexos.
- **Curva de aprendizado.** A adoção da computação serverless exige que os desenvolvedores aprendam novos conceitos, ferramentas e práticas recomendadas, o que pode adicionar complexidade ao processo de desenvolvimento.

Fontes:
https://www.ibm.com/br-pt/think/insights/microservices-advantages-disadvantages

https://www.ibm.com/br-pt/think/topics/serverless