UNIVERSIDADE FEDERAL DE SERGIPE

CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA

DEPARTAMENTO DE COMPUTAÇÃO

Características, boas práticas e soluções no desenvolvimento de aplicações com arquitetura de microserviços

Trabalho de Conclusão de Curso

João Paulo Feitosa Secundo

São Cristóvão – Sergipe

2022

UNIVERSIDADE FEDERAL DE SERGIPE

CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA

DEPARTAMENTO DE COMPUTAÇÃO

João Paulo Feitosa Secundo

Características, boas práticas e soluções no desenvolvimento de aplicações com arquitetura de microserviços

Trabalho de Conclusão de Curso submetido ao Departamento de Computação da Universidade Federal de Sergipe como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação.

Orientador(a): Rafael Oliveira Vasconcelos

São Cristóvão – Sergipe

2022

Lista de abreviaturas e siglas

API	Application Programing Interface - Interface para programação de aplicação
HTTP	HyperText Transfer Protocol - Protocolo de Transferência de HiperTexto
AMS	Arquitetura de MicroServiços
DoS	Denial of Service - Negação de serviço

Sumário

1
Introdução

Todos que têm contato com o ramo do desenvolvimento de software provavelmente já ouviu o termo *SaaS (Software as a Service)*, ou software como um serviço. Mas ao contrário do que alguns pensam, essa expressão é mais do que apenas um modelo de negócio.

O crescimento da internet e a onipresença da computação móvel tem mudado o jeito como software é desenvolvido nos últimos tempos. A tendência que tem-se observado é a de oferecer software não mais como um pacote completo e fechado, mas sim como um pacote flexível e em constante melhoria, o que implica na mudança do foco dos desenvolvedores para construir componentes leves e auto-contidos, que permitam que mudanças sejam desenvolvidas e implantadas rápida e independetemente. A partir disso originou-se um novo paradigma de desenvolvimento, chamado de "microsserviços". (Middleware Lab, 2021).

> Se você quiser projetar um aplicativo que seja multilíngue, facilmente escalável, fácil de manter e implantar, altamente disponível e que minimize falhas, use a arquitetura microservices para projetar e implantar um aplicativo em nuvem. (Oracle Corporation, 2021)

Esse estilo de arquitetura de software é amplamente considerado a melhor maneira de estruturar um sistema de software como um serviço. (XU et al., 2016)

Nesse trabalho serão discutidos as características da arquitetura de microsserviços, as boas práticas que devem ser seguidas no desenvolvimento de aplicações com essa arquitetura, e as (soluções ou ferramentas?) mais usadas.

1.1 Objetivos

1.1.1 Objetivo geral

Discutir, em alto nível, a arquitetura de microsserviços e suas características. Analisar as boas práticas e soluções mais usadas no desenvolvimento de aplicações que utilizam essa arquitetura. Modelar e implementar um exemplo de aplicação usando a arquitetura de microsserviços.

1.1.2 Objetivos específicos

- Caracterizar a arquitetura de microsserviços;

- Reunir boas práticas usadas na implementação de aplicações com arquitetura de microsserviços;

- Reunir ferramentas usadas na implementação de aplicações com arquitetura de microsserviços;

- Propor ideias e passos para como migrar do monolito para os microsserviços;

Analisar a eficiência dessas boas práticas;

Analisar a eficiência dessas ferramentas;

- Propor uma combinação das boas práticas e das ferramentas para o desenvolvimento de aplicações com arquitetura de microsserviços.

1.2 Metodologia

Para caracterizar a arquitetura de microsserviços, foram pesquisados as seguintes termos nas bases científicas ScienceDirect, SpringerLink e GoogleScholar:

- (microservices or microservice) and pattern

- (microservices or microservice) and provision

A partir dos principais resultados obtidos com essas buscas e os trabalhos em respectivas referências bibliográficas, foram extraídas as características que todo microsserviço deve ter.

(metodologia quanto aos objetivos, quanto a execução. passo a passo que vai seguir durante o trabalho. explicar como analisar/testar essa eficiencia (objetivos especificos))

2

Fundamentação teórica

Este capítulo aprensenta uma introdução sobre as arquiteturas de monolito e de microsserviços, e investiga trabalhos relacionados.

2.1 As aplicações monolíticas

Aplicações monolíticas, também chamadas de monólitos, são aplicações que possuem as camadas de acesso aos dados, de regras de negócios, e de interface de usuário em um único programa em uma única plataforma. Os monólitos são autocontidos e totalmente independentes de outras aplicações. Eles são feitos não para uma tarefa em particular, mas sim para serem responsáveis por todo o processo para completar determinada função. Em outras palavras, as aplicações monolíticas têm problema de modularidade. Elas podem ser organizadas das mais variadas formas e fazer uso de padrões arquiteturais, mas são limitadas em muitos outros aspectos, citados na subseção 2.1.2.

2.1.1 Benefícios

O maior e melhor benefício da arquitetura monolítica é sua simplicidade. Uma aplicação simples é uma aplicação fácilmente entendida pelos seus desenvolvedores, o que melhora sua manutenibilidade. Para aplicações com um domínio simples, como um e-commerce de calçados por exemplo, optar por uma arquitetura complexa como a de microsserviços significaria adicionar uma enorme complexidade - provavelmente desnecessária - em seu desenvolvimento e infraestrutura.

Outra vantagem dos monolitos é sua facilidade de construção, tanto em relação a sua infraestrutura quanto ao seu desenvolvimento. Dentre todos os tipos de arquitetura, os monólitos têm o tipo de infraestrutura mais fácil de se construir, e além disso, neles geralmente não é necessário haver comunicação entre diferentes serviços ou máquinas, então os desenvolvedores não precisarão se preocupar com a complexidade que acompanha essa comunicação.

Até certo tamanho, são fáceis de manter porque são fáceis de serem entendidos. Porém, depois de crescer excessivamente, um monólito pode se tornar um emaranhado complexo de funcionalidades que são difíceis de diferenciar, de separar, e de manter. E então começam a surgir as limitações deles...

2.1.2 Limitações

As limitações das aplicações monolíticas incluem:

Crescimento, velocidade de desenvolvimento, e manutenção - Depois de chegar num certo tamanho, torna-se muito difícil desenvolver funcionalidades novas, ou mesmo prover manutenção às já existentes. Padrões de organização podem amenizar a situação, mas não eliminam o problema.

Confiabilidade -

Escalabilidade -

Reutilização -

Implantação - Necessidade de compilar toda a aplicação, mesmo as partes em que não houve mudanças, a cada implantação.

Resiliência - Falhas relativamente pequenas podem prejudicar toda a aplicação, mesmo as partes que não tiveram relação com a falha.

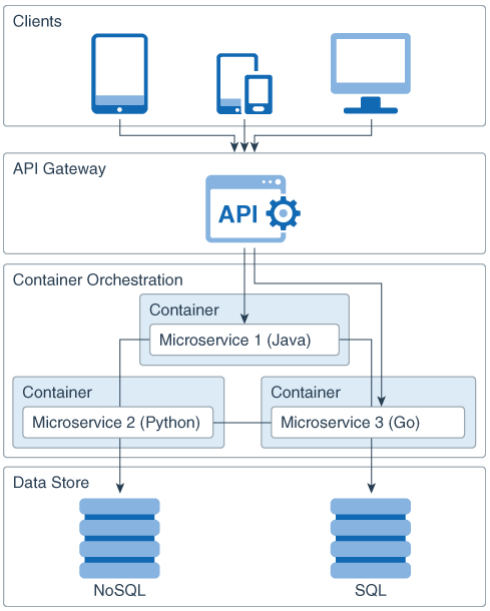
Flexibilidade - As escolhas de tecnologias são mais limitadas. Um projeto tende a usar apenas 1 solução devido a problemas de complexidade ou compatibilidade que podem surgir ao usar mais.

2.2 Os microsserviços

Microsserviços é uma abordagem de arquitetura de software. Aplicações com uma arquitetura de microsserviços são separadas em partes, chamadas de microsserviços, que são classificadas e se comunicam por meio de uma rede. Microsserviços oferecem capacidades de negócio (funcionalidades relacionadas às regras de negócio da aplicação) ou capacidades de plataforma (funcionalidades relacionadas ao ambiente de execução da aplicação), tratando um aspecto em particular da aplicação. Eles se comunicam por meio de APIs bem definidas, contratos de dados, e configurações. O "micro"em microsserviços faz referência não ao tamanho do serviço, mas sim ao seu escopo de funcionalidade. Eles oferecem apenas uma determinada funcionalidade, tornando-se especialistas nela. Assim sendo, microsserviços não necessariamente devem ser pequenos em tamanho, mas fazem apenas uma tarefa e a fazem eficientemente. (FAMILIAR, 2015)

Sendo especialistas em apenas uma tarefa, microsserviços têm características e comporta-

Figura 1 – Aplicação com arquitetura de microsserviços



Fonte: Oracle Corporation (2021)

mentos que os diferenciam de outras arquiteturas orientadas a serviços, os quais serão discutidos no Capítulo 3.

A Figura 1 exemplifica uma aplicação com arquitetura de microserviços. Inicialmente os usuários da aplicação (camada *Clients*) fazem requisições à api para obter as informações desejadas. O *API Gateway* - que é reponsável por integrar os serviços e será melhor discutido no Capítulo 4 - fará as devidas requisições para os devidos microsserviços (localizados na camada *Container Orchestration*). Esses microsserviços então buscarão a informação necessária no devido banco de dados (camada *Data Store*).

2.3 Trabalhos relacionados

2.3.1 Microservices, IoT and Azure, por Bob Familiar - capítulo 2: What is a microservice

O capítulo 2 do livro de Bob Familiar descreve o que é um microsserviço, suas características e implicações, benefícios, e desafios.

"Microservices do one thing and they do it well". Como é explicado por Familiar (2015) , microsserviços representam business capabilities definidos usando o design orientado a domínio, são testados a cada passo do *pipeline* de implantação, e lançados por meio de automação, como serviços independentes, isolados, altamente escaláveis e resilientes em uma infraestrutura em núvem distribuída. Pertecem a um time único de desenvolvedores, que trata o desenvolvimento

do microsserviço como um produto, entregando software de alta qualidade em um processo rápido e iterativo com envolvimento do cliente e satisfação como métrica de sucesso.

2.3.2 A Systematic Mapping Study on Microservices Architecture in DevOps, por Waseem, M., Liang, P. e Shahin, M.

Esse trabalho tem o objetivo de sistematicamente identificar, analisar, e classificar a literatura sobre microsserviços em DevOps.

Inicialmente o leitor é contextualizado no mundo dos microsserviços e a cultura DevOps. Os autores usam a metodologia de pesquisa de um estudo de mapeamento sistemático da literatura publicada entre Janeiro de 2009 e Julho de 2018. Após selecionados 47 estudos, é feita a classificação deles de acordo com os critérios definidos pelos autores, e então é feita a discussão sobre os resultados obtidos - são expostos a quantidade de estudos sobre determinados tópicos em microsserviços, problemas e soluções, desafios, métodos de descrição, design patterns, benefícios, suporte a ferramentas, domínios, e implicações para pesquisadores e praticantes.

Os principais resultados são: (1) São identificados Três temas de pesquisa em AMS com DevOps “desenvolvimento e operações de microsserviços em DevOps”, “abordagens e suporte a ferramentas para sistemas baseados em AMS em DevOps”, e “Experiência de migração de AMS em DevOps”. (2) São identificados 24 problemas e apontadas suas respectivas soluções com respeito a implementação de microsserviços com DevOps. (3) A AMS é descrita principalmente usando caixas e linhas. (4) A maioria das qualidades da AMS são afetadas positivamente quando aplicadas com DevOps. (5) 50 ferramentas que suportam a construção de sistemas baseados em AMS são apontados. (6) A combinação da AMS e DevOps tem sido aplicada em uma ampla variedade de domínios de aplicações.

(Comparar cada trabalho com o meu trabalho. Coisas que eles não abordam e que eu abordo)

3
Características

3.1 Propriedades

3.1.1 Autonomia e Isolamento

Autonomia e isolamento significa que microsserviços são unidades auto-contidas de funcionalidade com dependências de outros serviços fracamente acopladas e são projetados, desenvolvidos, testados e lançados independentemente. (FAMILIAR, 2015). O termo autônomo pode ser definido como - existe ou é capaz de existir independetemente das outras partes. O termo isolado, como - separado das outras partes.

3.1.2 Elasticidade, resiliência, e responsividade

Microsserviços são reusados entre muitas soluções diferentes e portanto devem ser escaláveis de acordo com o uso. Devem ser resilientes, isso é, ser tolerantes a falhas e ter um tempo de recuperação razoável quando algo der errado. Além disso, devem ser responsivos, tendo um desempenho razoável de acordo com o uso. (FAMILIAR, 2015) O termo elástico pode ser definido como - capaz de retornar ao tamanho/formato original depois de ser esticado, comprimido ou expandido. O termo resiliente, como - resistente às mudanças negativas. O termo responsivo, como - Rápido em responder e reagir.

3.1.3 Orientação-a-mensagens e programabilidade

Microsserviços dependem de APIs e contratos de dados para definir como interagir com o serviço. A API define um conjunto de endpoints acessíveis por rede, e o contrato de dados define a estrutura da mensagem que é enviada ou retornada. (FAMILIAR, 2015). O termo orientado-a-mensagens pode ser definido como - Software que conecta sistemas separados em

uma rede, carregando e distribuindo mensagens entre eles. O termo programável, como - Obedece a um plano de tarefas que são executadas para alcançar um objetivo específico.

3.1.4 Configurabilidade

Microsserviços devem provêr mais do que apenas uma API e um contrato de dados. Para que seja reusável e para que possa resolver as necessidades do sistema que o use, cada microsserviço tem níveis diferentes de configuração, e esta configuração pode ser feita de diferentes formas. (FAMILIAR, 2015). O termo configurável pode ser definido como - Projetado ou adaptado para formar uma configuração ou para algum propósito.

3.1.5 Automação

O ciclo de vida de um microsserviço deve ser totalmente automatizado, desde o design até a implantação. O termo automatizado pode ser definido como - Funcionar sem precisar ser controlado diretamente.

3.2 Vantagens

3.2.1 Evolução

Quanto maior e mais antigo o software, mais difícil é de dar manutenção, e monolitos envelhecem com maior velocidade do que microsserviços. Entretanto, é possível migrar de um sistema monolítico para a arquitetura de microsserviços aos poucos, um serviço por vez, identificando capacidades de negócio, implementando-as como um microsserviço, e integrando com uso de padrões de baixo acoplamento. Ao longo do tempo, mais e mais funcionalidades podem ser separadas e implementadas como microsserviço, até que o núcleo da aplicação monolítica se transforme em apenas um outro serviço, ou um microsserviço. (FAMILIAR, 2015)

3.2.2 Possibilidade de uso de diferentes ferramentas

Cada microsserviço disponibiliza suas funcionalidades por meio de APIs e contratos de dados em uma rede. A comunicação independe da arquitetura que o microsserviço faz uso, então cada microsserviço pode escolher seu sistema operacional, linguagem e banco de dados. Isso é especialmente valioso para times com dificuldade de comunicação, pois cada time precisa apenas de conhecimento da arquitetura do microsserviço em que trabalha. (FAMILIAR, 2015)

3.2.3 Alta velocidade

Com um time responsável por cuidar do ciclo de desenvolvimento e sua automação, a velocidade com que microsserviços podem ser desenvolvidos é muito maior do que fazer o

equivalente para uma solução monolítica. (FAMILIAR, 2015)

3.2.4 Reusável e combinável

Microsserviços são reusáveis por natureza. Eles são entidades independentes que provêm funcionalidades em um determinado escopo por meio de padrões de internet aberta. Para criar soluções para o usuário final, multiplos microsserviços podem ser combinados. (FAMILIAR, 2015)

3.2.5 Flexibilidade no ambiente de execução

A implantação de microsserviços é altamente dependente de sua automação. Para garantir flexibilidade de ambiente de execução, essa automação pode incluir configuração de cenários diferentes de uso, não apenas para produção, mas também para desenvolvimento e testagem, possibilitando que o microsserviço tenha o melhor desempenho em diversos cenários. Para tanto, é necessário o uso de ferramentas que configurem essa flexibilidade. (FAMILIAR, 2015). Tais ferramentas serão melhor discutidas no Capítulo 5.

3.2.6 Flexibilidade na escolha de tecnologias

Cada microsserviço pode ser desenvolvido usando uma linguagem de programação e estrutura que melhor se adapte ao problema que ele é projetado para resolver, o que oferece mais possibilidades de tecnologias para usar. (Oracle Corporation, 2021)

3.2.7 Versionável e Substituível

Com o controle completo dos cenários de implantação, é possível manter versões diferentes de um mesmo serviço rodando ao mesmo tempo, proporcionando retrocompatibilidade e fácil migração. Além disso, serviços podem ser atualizados ou mesmo substituidos sem ocasionar indisponibilidade do serviço. (FAMILIAR, 2015)

3.3 Desafios

However, developing applications in the microservices architecture presents three main challenges: (a) how to program systems that consists of a large number of services running in parallel and distributed over a cluster of computers; (b) how to reduce the communication overhead caused by executing a large number of small services; (c) how to support the flexible deployment of services to a network to achieve system load balance. (XU et al., 2016)

3.3.1 Comunicação

cross-platform compatibility issues and inconsistent call standards issues in the process of development and call microservices. (ZUO et al., 2020)

3.3.2 [re]Organizaçao

Organizar o sistema e o time para sustentar uma arquitetura de microsserviços é um grande desafio. Como explica Familiar (2015):

> If you are part of a command-and-control organization using a waterfall software project management approach, you will struggle because you are not oriented to high-velocity product development. If you lack a DevOps culture and there is no collaboration between development and operations to automate the deployment pipeline, you will struggle. (FAMILIAR, 2015)

3.3.3 Plataforma

Criar o ambiente de execução para microsserviços requer um grande investimento em infraestrutura dinâmica em *data centers* dispersos para garantir maior disponibilidade. Se sua atual plataforma *on-premises* não suporta automação, infraestrutura dinâmica, escalamento elástico e alta disponibilidade, deve-se considerar uma plataforma na núvem. (FAMILIAR, 2015). Mais sobre soluções na núvem será discutido no Capítulo 5.

3.4 Identificação

Domain-driven design (projeto orientado a domínio) é uma técnica bem consolidada e muito usada no desenvolvimento de software. Entretanto, para aplica-la em microsserviços, é necessário analisar onde cada peça desse padrão de projeto deve ficar. Em vez de projetar os modelos e os contextos limitados separando-os em camadas, deve-se juntar os contextos com seus respectivos modelos, e procurar por possíveis pontos de separação da aplicação - um lugar onde a linguagem muda, por exemplo. Isso resultaria em um ponto de partida para separar as partes e formar uma arquitetura de microsserviços. (FAMILIAR, 2015)

esse parágrafo não encaixa nas caracteristicas - desafios ?...

3.5 Testes

Assim como em qualquer aplicação, o teste é uma parte crucial do seu desenvolvimento. Escrever e testar código não muda muito entre as arquiteturas monolítica e de microsserviços, contudo, nos microserviços existem mais testes a serem executados. Não deve-se testar o microsserviço apenas antes de seu lançamento, mas sim em cada passo do *pipeline* de implantação,

sempre automatizando o máximo de etapas possível, para assim garantir uma entrega rápida de software de qualidade. (FAMILIAR, 2015)

3.6 Descoberta

Encontrar microsserviços em um ambiente distribuido pode ser feito de algumas maneiras diferentes. A informação pode ser armazenada diretamente no código, pode ser guardada e acessada em um arquivo, ou pode ser construido um microsserviço para encontrar outros microsserviços e disponibilizar suas localizações. Contudo, para prover detectabilidade como um serviço será necessário adquirir um produto de terceiros, integrar um projeto aberto, ou desenvolver sua própria solução. (FAMILIAR, 2015)

4

Boas práticas

4.1 Antes de tudo, comece pelo monólito

> But as with any architectural decision there are trade-offs. In particular with microservices there are serious consequences for operations, who now have to handle an ecosystem of small services rather than a single, well-defined monolith. Consequently if you don't have certain baseline competencies, you shouldn't consider using the microservice style. (FOWLER, 2014)

Fowler (2014) afirma que existem 3 pré-requisitos para se adotar uma arquitetura de microserviços, e que na grande maioria dos casos, deve-se começar pela arquitetura monolítica até que o sistema já esteja bem definido. Os pré-requisitos são - provisionamento rápido, monitoramento básico e deploy rápido de aplicação.

4.1.1 Provisionamento rápido

No contexto da computação, provisionamento significa disponibilizar um recurso, como uma máquina virtual por exemplo. Para produzir software, é necessário provisionar muitos recursos, tanto para os desenvolvedores quanto para o cliente. Naturalmente, o provisionamento é mais fácil na núvem. Na AWS por exemplo, para conseguir uma nova máquina, basta lançar uma nova instância e acessá-la - um processo muito rápido quando comparado ao *on-premises*, onde precisaria-se comprar uma nova máquina, esperar chegar, configurá-la, e só então ela estará pronta. Para alcançar um provisionamento rápido, será necessário bastante automação.

4.1.2 Monitoramento básico

Muitas coisas podem dar errado em qualquer tipo de arquitetura, mas em especial nos microserviços pois cada serviço é fracamente acoplado, estando sujeitos não só a falhas no código, mas também na comunicação, na conexão, ou até falhas físicas. Portanto o monitoramento é

crucial nesse tipo de arquitetura para que problemas, especialmente os mais graves possam ser detectados no menor tempo possível. Além disso, o monitoramento também pode ser usado para detectar problemas de negócio, como uma redução nos pedidos por exemplo.

4.1.3 Implantação rápida

Na arquitetura de microserviços a implantação geralmente é feita separadamente para cada microserviço. Com muitos serviços para gerenciar, ela pode se tornar uma tarefa árdua, portanto será novamente necessário uma automação dessa etapa, que geralmente envolve um *pipeline* de implantação, que deve ser automatizado o máximo possível.

4.2 Configuração

4.3 Implantação

4.4 Comunicação entre microserviços

4.5 APIs

Considerando que APIs são uma parte crucial no desenvolvimento de microserviços, sendo responsável por grande parte da comunicação que se faz necessária para conectar tantos serviços separados e manter um funcionamento eficiente e livre de falhas, esse trabalho terá um foco grande em boas práticas no desenvolvimento de APIs.

4.5.1 Segurança em APIs

Autenticação

Incluir autenticação em uma API consiste em exigir uma prova de autorização do uso da API. A autenticação nas APIs é indispensável para aumentar a segurança, e existem formas diferentes de implementá-la, as quais serão melhor discutidas no Capítulo 5.

Validação de entradas

Validar entradas significa verificar as requisições que chegam com o intuito de garantir que elas não contém dados impróprios, tais como injeções de SQL ou *scripting* entre sites (scripting significa executar uma determinada sequência de comandos). Essa validação deve ser implementada tanto em nível sintático como em semântico, isso é, tanto impondo correção da sintaxe quanto impondo correção de valores. (RAPIDAPI, 2022a)

API Gateway

Um API Gateway funciona como uma porta única de entrada para as APIs de cada serviço. As requisições dos clientes são direcionadas para um único endereço, facilitando a organização das chamadas. Esse gateway fica situado entre o cliente e os serviços do *backend*, e é responsável por redirecionar as requisições recebidas para os serviços apropriados, assim o gerenciamento das chamadas pode ser feita em apenas um lugar em vez de em cada API de cada serviço. Além disso, pode-se implementar uma camada de segurança e de monitoramento. (RAPIDAPI, 2022a)

Limitação de taxa de requisições

Limitar a taxa de requisições é um jeito de proteger a infraestrutura do servidor nos casos de acontecerem grandes fluxos de requisições, tal como em um ataque de *DoS* (negação de serviço). Clientes terão seu acesso bloqueado caso enviem uma quantidade de requisições acima do limite determinado. (RAPIDAPI, 2022a)

Compartilhar o mínimo possível

Compartilhar o mínimo possível é uma medida de segurança genérica que deve ser adotada em tudo. Mais especificamente nas APIs, deve-se retornar estritamente apenas os dados necessários para o cliente. Muitas ferramentas usadas para implementar APIs incluem por padrão informações como se fossem marcas d'água, mas que podem ser removidas, tal como headers "X-Powered-By", que vazam informações do servidor que podem auxiliar usuários mal-intencionados. (RAPIDAPI, 2022a)

4.5.2 Testes em APIs

API testing is performed to test whether a particular API meets pre-defined parameters or not.

API testing includes testing APIs in isolation to ascertain if they meet the functionality, reliability, latency, performance, security, and other essential parameters.

How to test an API?

You can code the test or use cURL for API testing.

RapidAPI offers RapidAPI Client for VS Code to test APIs locally inside Visual Studio Code. You can also schedule API tests using RapidAPI Studio.

Why should you perform API testing?

- Testing your APIs timely helps to ensure your app is up all the time.

- It helps to detect API security and performance issues.

Benefits of API Testing

- When API tests fail, you will know precisely where the issue lies that crashed the system.

- As data is exchanged via XML or JSON, you can write API tests in your preferred language.

- API testing also helps to release the next API version faster. (RAPIDAPI, 2022b)

4.5.3 Otimizações em API

Cache response - Caching avoids excessive database queries. For endpoints that frequently return the same response, caching can be implemented to reduce the number of calls to the API and improve performance.

Compress data - The transfer of large payloads will slow down an API. Data compression combats this issue by decreasing the data size and improving speed. There are various compression methods available, a common one being GZIP.

Prevent over and under-fetching - Over-fetching results in unnecessary and unusable data, and under-fetching results in an incomplete response. Good architecture, planning, and appropriate API management tools are essential to avoid these.

Paginate and filter - Both pagination and filtering are great methods to reduce response complexity and improve user experience. Pagination enables the separation and categorization of data, and filtering limits the results of parameters.

Use PATCH not PUT - The PATCH and PUT methods are similar, but PATCH has performance advantages. When modifying a resource, PUT updates the entire resource, which is often unnecessary, whereas PATCH only updates a specific part. Therefore PATCH has a smaller payload. (RAPIDAPI, 2022c)

4.6 Testes

Mas além dos métodos mais conhecidos de testes, como test-driven development, teste de unidade e teste funcional, é necessário testar os microsserviços conforme passam pelo *pipeline* de implantação.

- Testes internos: Testar as funções internas do serviço, inclusive uso de acesso de dados, e caching.

- Teste de serviço: Testar a a implementação de serviço da API. Essa é uma implementação privada da API e seus modelos associados.

- Teste de protocolo: Testar o serviço no nível de protocolo, chamando a API sobre o determinado protocolo (geralmente HTTP).

- Composition Testing: Test the service in collaboration with other services within the context of a solution.

- Scalability/Throughput Testing: Test the scalability and elasticity of the deployed microservice.

- Failover/Fault Tolerance Testing: Test the ability of the microservice to recover after a failure.

- PEN Testing: Work with a third-party software security firm to perform penetration testing. NOTE: This will requires cooperation with Microsoft if you are pen testing microservices deployed to Azure.

4.7 A metodologia de 12 fatores

Asdf. (Oracle Corporation, 2021).

Qwer. (WIGGINS, 2017)

4.8 Do monólito aos microserviços

4.8.1 Identificação

If you are currently working with a complex layered architecture and have a reasonable domain model defined, the domain model will provide a roadmap to an evolutionary approach to migrating to a microservice architecture. If a domain model does not exist, you can apply domain-driven design in reverse to identify the bounded contexts, the capabilities within the system. (FAMILIAR, 2015)

4.8.2 Organização

Em uma mudança do monolito para microsserviços, é recomendado que não sejam feitas mudanças grandes e abruptas na sua organização. Em vez disso, deve-se procurar uma oportunidade com uma iniciativa de negócio para testar a fórmula proposta por Familiar (2015) :

- Formar um pequeno time inderdisciplinar (cross-functional?).

- Oferecer treinamento e orientação na adoção de práticas ágeis, como o scrum.

- Oferecer uma localização física separada para esse time trabalhar a fim de não afetá-lo negativamente por politicas internas ou hábitos antigos.

- Adotar uma abordagem de minimo produto viável para entregar pequenos mas incrementais *releases* de software, usando essa abordagem durante todo o ciclo de vida.

- Integrar esse serviço com sistemas existentes, usando um acoplamento solto.

- Percorrer esse ciclo de vida do microsserviço diversas vezes, fazendo as adaptações necessárias até chegar a equipe ficar confortável com o processo.

- Colocar o time principal em posições de liderança enquanto são formados novos times interdisciplinares para disseminar o conhecimento e a prática.

5

Soluções: Ferramentas

5.1 Flexibilidade

(tais como as ferramentas de Auto Scaling da AWS)

5.2 Plataforma

Microsoft Azure is a microservice platform, and it provides a fully automated dynamic infrastructure, SDKs, and runtime containers along with a large portfolio of existing microservices that you can leverage, such as DocumentDb, Redis In-Memory Cache, and Service Bus, to build your own microservices catalog. (FAMILIAR, 2015)

5.3 APIs

5.3.1 GraphQL

A query language for your API

GraphQL is a query language for APIs and a runtime for fulfilling those queries with your existing data. GraphQL provides a complete and understandable description of the data in your API, gives clients the power to ask for exactly what they need and nothing more, makes it easier to evolve APIs over time, and enables powerful developer tools. (The GraphQL Foundation, 2018)

5.3.2 API Gateway

Amazon API Gateway <https://aws.amazon.com/pt/api-gateway/?nc1=h_ls>

5.3.3 Ferramentas para segurança em APIs

Autenticação - Always use secure authentication methods such as OAuth, JWTs, or API Keys. It's not recommended to use basic HTTP authentication as it sends user credentials with each request. It is considered the least secure method.

Validação de entradas - Métodos de validação de entrada: JSON and XML Schema validation; Regular expressions; Data type validators available in framework; Minimum and maximum value range check for numerical inputs; Minimum and maximum length check for strings.

Usar um API Gateway - API Gateways are an all-in-one way to implement security, monitoring, and overall API management. They are a single entry point for API calls. They sit between the clients and a number of backend services to handle calls appropriately.

5.3.3.1 Métodos de autenticação

API Keys are unique identifiers assigned to clients, which grant them access to an API. They are passed to the server with every request and authenticate the client. They also provide authorization and can be used to identify a user's individual access permissions. API Keys are long alphanumerical strings designed to be almost impossible to guess. They are passed to servers as a query parameter or in an HTTP request header.

OAuth is a powerful framework that uses tokens to give apps limited access to a user's data without needing the user's password. The tokens used are restricted and only allow access to data that the user specified for the particular app. It works by the user(client) first requesting authorization from the resource owner. The user is then given a unique access token from an authorization server used in each request to the resource server.

Basic HTTP authentication involves the client passing the user's username and password with every request. This is done using an HTTP Header. Basic HTTP authentication is generally considered the least secure. However, if you decide to use it, ensure you are using an HTTPS connection. If not, data is a risk of being leaked.

Ferramenta para rate limiting. (RAPIDAPI, 2022a)

5.3.4 Ferramentas completas

Uma solução para a sobrecarga na execução de tantos microserviços é a um ambiente de desenvolvimento integrado na linguagem CAOPLE (XU et al., 2016). Essa plataforma oferece grande controle sobre o deployment e testagem de microserviços.

6
Conclusão

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Plano de Continuidade

Analisar a eficiência desses padrões e práticas.

Analisar a eficiência dessas soluções e ferramentas.

Propor uma combinação desses padrões e dessas ferramentas para a construção de uma aplicação com arquitetura de microsserviços.

Referências

FAMILIAR, B. What is a microservice? In: ____. *Microservices, IoT, and Azure: Leveraging DevOps and Microservice Architecture to Deliver SaaS Solutions*. Berkeley, CA: Apress, 2015. p. 9–19. ISBN 978-1-4842-1275-2. Disponível em: <https://doi.org/10.1007/978-1-4842-1275-2_2>. Citado 9 vezes nas páginas 8, 9, 11, 12, 13, 14, 15, 20 e 22.

FOWLER, M. *bliki: MicroservicePrerequisites*. 2014. Blog do Martin Fowler. Disponível em: <https://martinfowler.com/bliki/MicroservicePrerequisites.html>. Acesso em: 06 Oct 2022. Citado na página 16.

Middleware Lab. *What are Microservices? How Microservices architecture works*. Middleware Lab, 2021. Disponível em: <https://middleware.io/blog/microservices-architecture/>. Citado na página 5.

Oracle Corporation. topic, *Learn about architecting microservices-based applications on Oracle Cloud*. Oracle Corporation, 2021. Disponível em: <https://docs.oracle.com/pt-br/solutions/learn-architect-microservice>. Citado 4 vezes nas páginas 5, 9, 13 e 20.

RAPIDAPI. topic, *API security best practices*. 2022. Disponível em: <https://twitter.com/Rapid_API/status/1583442979737346055>. Acesso em: 25 Oct 2022. Citado 3 vezes nas páginas 17, 18 e 23.

RAPIDAPI. topic, *API Testing. What is it?* 2022. Disponível em: <https://twitter.com/Rapid_API/status/1584530377686646784>. Acesso em: 25 Oct 2022. Citado na página 19.

RAPIDAPI. topic, *Optimize API performance with these 5 tips*. 2022. Disponível em: <https://twitter.com/Rapid_API/status/1583599981369303040>. Acesso em: 25 Oct 2022. Citado na página 19.

The GraphQL Foundation. Documentation, *GraphQL | A query language for your API*. 2018. Disponível em: <https://graphql.org/>. Acesso em: 26 Oct 2022. Citado na página 22.

WIGGINS, A. topic, *The Twelve-Factor App*. 2017. Disponível em: <https://12factor.net/>. Acesso em: 21 Oct 2022. Citado na página 20.

XU, C. et al. CAOPLE: A Programming Language for Microservices SaaS. In: *2016 IEEE Symposium on Service-Oriented System Engineering (SOSE)*. [S.l.: s.n.], 2016. p. 34–43. Citado 3 vezes nas páginas 5, 13 e 23.

ZUO, X. et al. An api gateway design strategy optimized for persistence and coupling. *Advances in Engineering Software*, v. 148, p. 102878, 2020. ISSN 0965-9978. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0965997820304452>. Citado na página 14.