UNIVERSIDADE FEDERAL DE SERGIPE

CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA

DEPARTAMENTO DE COMPUTAÇÃO

Padrões e soluções de 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

Padrões e soluções de 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

Sumário

1 Introdução

In recent years, the rise of the internet and the ubiquity of mobile computing have made it necessary for application developers to design their applications focusing on a lightweight, self-contained component.

Developers need to deploy applications quickly and make changes to the application without a complete redeployment. This has led to a new development paradigm called "microservices,"where an application is broken into a suite of small, independent units that perform their respective functions and communicate via APIs.

Although independent units, any number of these microservices may be pulled by the application to work together and achieve the desired results.

For the past several years, we have been developing standards and practices for team development of large, complex systems using a layered, monolithic architecture. This is reflected in how we organize into teams, structure our solutions and source code control systems, and package and release our software.

1.1 Objetivos

Esta seção descreve os objetivos do trabalho. Esta é a seção 1.1. Veja os objetivos específicos em subseção 1.1.2.

1.1.1 Objetivo geral

Analisar e resumir o estado da arte em desenvolvimento de aplicações com arquitetura de microserviços, e elaborar sobre os padrões e soluções mais encontrados. (tcc2): Modelar/implementar um exemplo de aplicação com arquitetura de microserviços.

1.1.2 Objetivos específicos

(TCC 1:)

- Resumir o estado da arte em desenvolvimento de aplicações com arquitetura de microserviços;

- Reunir padrões e práticas comuns na implementação de aplicações com arquitetura de microserviços;

- Reunir soluções e ferramentas usadas na implementação de aplicações com arquitetura de microserviços;

- (Propor um caminho de como migrar do monolito para os microserviços?)

(TCC 2:)

- Analisar/testar a eficiência desses padrões e práticas, por meio de (estudos de caso? análise da literatura? exemplos de empresas que as usam?);

- Analisar/testar a eficiência dessas soluções e ferramentas, por meio de (estudos de caso? análise da literatura? exemplos de empresas que as usam?);

- Propor uma combinação desses padrões e dessas ferramentas para a construção de um aplicativo com arquitetura de microserviços.

2

Fundamentação teórica

Introdução sobre as arquiteturas de monolito e de microserviços. Revisão da literatura.

2.1 Os monolitos

Aplicações monolíticas 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 monolitos 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 não têm modularidade. Elas podem ser organizadas das mais variadas formas, e fazer uso de padrões arquiteturais, mas são limitadas em muitos outros aspectos.

2.1.1 Benefícios

Simples. Fácil de construir. Até certo tamanho, é mais fácil de manter. (elaborar melhor)

2.1.2 Limitações

Crescimento, escalamento, manutenção, reutilização, flexibilidade (elaborar melhor)

This had several limitations such as inflexibility, lack of reliability, difficulty scaling, slow development, and so on. It was to bypass these issues that microservices architecture was created.

Alguns problemas da solução monolítica: - A necessidade de *buildar* toda a aplicação, mesmo as partes em que não houveram mudanças, a cada *deploy*. - Falhas relativamente pequenas podem prejudicar toda a aplicação, mesmo as partes que não tiveram relação com a falha. - As escolhas de tecnologias são mais limitadas. Um projeto **tende** a usar apenas 1 solução.

2.2 O que são microserviços

Aplicações com uma arquitetura de microserviços são separadas em partes pequenas, chamadas de microserviços, que são classificadas e se comunicam usando uma rede. Microserviços oferecem capacidades de negócio ou de plataforma, tratando um aspecto em particular da aplicação. Eles se comunicam por meio de APIs bem definidas, contratos de dados, e configurações. O "micro"em microserviços faz referência não ao tamanho do serviço, mas sim ao seu escopo de funcionalidade. Eles oferecem essa e apenas essa determinada funcionalidade. Assim sendo, microserviços não necessariamente devem ser pequenos em tamanho, mas fazem apenas uma tarefa e fazem isso bem.

No contexto de realizar uma tarefa bem, microserviços têm propriedades e comportamentos que os diferenciam de outras arquiteturas orientadas a serviços.

2.2.1 Autonomia e Isolamento

Autonomia e isolamento significa que microserviços são unidades auto-contidas de funcionalidade com dependências de outros serviços fracamente acopladas e são projetados, desenvolvidos, testados e lançados independentemente.

Autônomo - Existe ou é capaz de existir independetemente das outras partes.

Isolado - Separado das outras partes.

2.2.2 Elasticidade, resiliência, e responsividade

Microserviços são reusados entre muitas soluções diferentes e portanto devem ser escaláveis de acordo com o uso. Devem ser tolerantes a falhas e ter um tempo de recuperação razoável se algo der errado. E também devem ser responsivos, tendo um desempenho razoável de acordo com o uso.

Elástico: Capaz de retornar ao tamanho/formato original depois de ser esticado, comprimido ou expandido.

Resiliente: Resistente a mudanças ruins.

Responsivo: Rápido em responder e reagir.

2.2.3 Orientação-a-mensagens e programabilidade

Microserviços 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.

Orientado-a-mensagens: Software que conecta sistemas separados em uma rede, carregando e distribuindo mensagens entre eles.

Programável: Obedece a um plano de tarefas que são executadas para alcançar um objetivo específico.

2.2.4 Configurabilidade

Microserviços devem provêr mais do que apenas uma API e um contrato de dados. Para que seja reusável e para que possa resolver as necessidades de que sistema que o use, cada microserviço tem níveis diferentes de configuração, e esta configuração pode ser feita de diferentes formas.

Configurável: Projetado ou adaptado para formar uma configuração ou para algum propósito.

2.2.5 Automação

O ciclo de vida de um microserviço é totalmente automatizado, desde o design até a implantação.

Automatizados: Funcionar sem precisar ser controlado diretamente.

2.3 Vantagens da arquitetura de microserviços

2.3.1 Evolução

Quanto maior e mais antigo o software, mais difícil é de dar manutenção, e monolitos envelhecem com maior velocidade do que microserviços. Mas é possível migrar de um sistema monolito para a arquitetura de microserviços aos poucos, um serviço por vez, identificando (capacidades/funcionalidades/escopos) de negócio, implementando-as como um microserviço, e integrando com uso de padrões de acoplamento solto [relaxado?folgado?] (loose coupling). Ao longo do tempo, mais e mais funcionalidades podem ser migradas, até que o monolito se transforme em apenas um outro serviço, ou um microserviço

2.3.2 Possibilidade de uso de diferentes ferramentas

Cada microserviço disponibiliza suas funcionalidades por meio de APIs e contratos de dados em uma rede. A comunicação independe da arquitetura do microserviço faz uso, então cada um pode escolher seu sistema operacional, linguagem e banco de dados.

Isso é especialmente valioso para times multinacionais, pois cada time precisa apenas de conhecimento da arquitetura do microserviço em que trabalha.

2.3.3 Alta velocidade

Com um time responsável por cuidar do ciclo de desenvolvimento e sua automação, a velocidade com que microserviços podem ser desenvolvidos é muito maior do que fazer o equivalente para uma solução monolítica.

2.3.4 Reusável e combinável

Microserviços são reusáveis por natureza. Eles são entidades independentes que provêm funcionalidades em um determinado escopo por meio de (open internet standards). Para criar soluções para o usuário final, multiplos microserviços podem ser combinados.

2.3.5 Flexível

A implantação de microserviços é definida por sua automação. Essa automação pode incluir configuração de cenários diferentes de uso, não apenas para produção, mas também para desenvolvimento e testagem, possibiltando que o microserviço tenha o melhor desempenho em diversos cenários. Para tanto é necessário uso de ferramentas que configurem essa flexibilidade. (tais como as ferramentas de Auto Scaling da AWS)

2.3.6 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 substituidos sem causar tempo indisponível.

2.4 Desafios

2.4.1 [re]Organizaçao

Organizar o sistema e o time para sustentar uma arquitetura de microserviços é um grande desafio. *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*.

Em uma mudança do monolito para microserviços, 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) :

• Form a small cross-functional team.

• Provide training and guidance on adopting Agile, Scrum, Azure, and microservice architecture.

• Provide a separate physical location for this team to work so that they are not adversely effected by internal politics and old habits.

• Take a minimal-viable-product approach and begin to deliver small incremental releases of one microservice, taking the process all the way through the lifecycle.

• Integrate this service with the existing systems using a loosely coupled approach.

• Go through the lifecycle on this microservice several times until you feel comfortable with the process.

• Put the core team into leadership positions as you form new cross-functional teams to disseminate the knowledge.

2.4.2 Plataforma

Creating the runtime environment for microservices requires a significant investment in dynamic infrastructure across regionally disperse data centers. If your current on-premises application platform does not support automation, dynamic infrastructure, elastic scale, and high availability, then it makes sense to consider a cloud platform. 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.

2.5 Identificação

Domain-driven design (design orientado a domínio) é uma tecnica bem consolidada e muito usada. Mas para aplica-la em microserviços, é preciso analisar onde cada peça deve ficar. Em vez de modelar os modelos e os (contextos limitados) separando-os em camadas, pode-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 para uma arquitetura de microserviço.

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.

2.6 Testes

Assim como a automação, testar o microserviço em cada passo do *pipeline* de *deploy* é necessário para a entrega rápida de software de qualidade.

Escrever e testar código não muda muito entre as arquiteturas monolítica e de microserviços. Mas além dos métodos mais conhecidos de testes, como test-driven development, teste de unidade e teste funcional, é necessário testar os microserviços conforme passam pelo *pipeline* de *deploy*.

- Internals Testing: Test the internal functions of the service including use of data access, caching, and other cross-cutting concerns.

- Service Testing: Test the service implementation of the API. This is a private internal implementation of the API and its associated models.

- Protocol Testing: Test the service at the protocol level, calling the API over the specified wire protocol, usually HTTP(s).

- 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.

2.7 Detectável

Encontrar microserviços em um ambiente distribuido pode ser feito de algumas maneiras diferentes: Hardcode no código, guardar em um arquivo, ou fazer um microserviço para encontrar outros microserviços e disponibilizar suas localizações. Para prover detectabilidade como um serviço será necessário adquirir um produto de terceiros, integrar um projeto aberto, ou desenvolver sua própria solução.

2.8 Trabalhos relacionados

"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 microserviço, suas características e implicações, benefícios, e desafios.

"Microservices do one thing and they do it well". Como é explicado por Familiar (2015) , microserviços representam business capabilities definidos usando o design orientado a domínio, são testados a cada passo do *pipeline* de *deploy*, 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 microserviço como um produto, entregando software de alta qualidade em um processo rápido e iterativo com envolvimento do cliente e satisfação como métrica de sucesso.

"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 microserviços em DevOps.

Inicialmente o leitor é contextualizado no mundo dos microserviços 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 microserviços, problemas e soluções, desafios, métodos de descrição, design patterns, benefícios, suporte a ferramentas, domínios, e implicações para pesquisadores e praticantes.

the key results are: (1) Three themes on the research on MSA in DevOps are “microservices development and operations in DevOps”, “approaches and tool support for MSA based systems in DevOps”, and “MSA migration experiences in DevOps”. (2) 24 problems with their solutions regarding implementing MSA in DevOps are identified. (3) MSA is mainly described by using boxes and lines. (4) Most of the quality attributes are positively affected when employing MSA in DevOps. (5) 50 tools that support building MSA based systems in DevOps are collected. (6) The combination of MSA and DevOps has been applied in a wide range of application domains. Conclusions: The results and findings will benefit researchers and practitioners to conduct further research and bring more dedicated solutions for the issues of MSA in DevOps.

3

Desenvolvimento

Provisionamento

Lançamento

Comunicação entre serviços

APIs

API gateway

3.1 Problemas, soluções e desafios.

3.1.1 Antes de tudo, o monolito

> 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.

3.1.1.1 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.

3.1.1.2 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.

3.1.1.3 Deploy rápido de aplicação

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

3.1.2 Configuração

3.1.3 Deploy

3.1.4 Comunicação entre microserviços

Requisitos de sistemas baseados em AMS

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.

This category reports the problems and solutions related to the requirements of MSA based systems.

Study (S33) proposes the VM autoconfiguration methodology to address performance issues; VM auto-configuration method creates the central domain control agent for optimizing the performance of MSA based systems.

Study (S18) proposes Unicorn framework to avoid delays and network performance issues, whereas Study (S24) suggests that architects should try not to decompose microservices too fine-grain.

Study (S16) presents a DevOps based approach called “Neo-Metropolis”. This approach offers open source solutions (e.g., Terraform, Ansible, Mesos, and Hadoop) to deal with scalability

and elasticity of MSA based systems across different cloud platforms. Study (S18) argues for the use of containers to deal with scalability issues because containers provide an easy way to scale operations by creating more copies of the services (S18). Study (S41) suggests that developing microservices around business capabilities can address this scalability issue.

3.1.5 Design of MSA based systems in DevOps

This category reports the problems and solutions related to the design of MSA based systems in DevOps (see Figure 6), which can be further classified into application decomposition (S28, S33, S35, S37), security and privacy (S10, S18, S20, S36), and uncertainty (S01). Study (S28) recommends the Domain-Driven Design (DDD) pattern to address the application decomposition problem. By applying the DDD pattern, architects identify the bounded context (capabilities within the system) that can be used as a starting point for defining microservices. Similarly, Study (S33) recommends the Model-ViewController (MVC) pattern for application decomposition into microservices in terms of business scope, functionalities, and responsibilities. Study (S10) presents the DevOps based ARCADIA framework to address security issues. This framework enables security and privacy across the microservices development lifecycle by providing multi-vendor security solutions (e.g., FWaaS and OAuth 2). Study (S18) presents DevOps based Unicorn framework that offers policies and constraints to meet security requirements of MSA based systems, whereas Study (S36) suggests that a combination of standard cryptographic primitives (e.g., hash and MAC functions for authentication encryption) can provide a high level of security to microservices communication and flexible authentication to DevOps teams. To deal with uncertainty issues in cloud-native architecture, Study (S01) proposes the theory-based control models at runtime patterns. Models at runtime patterns address the uncertainty aspects (e.g., resource availability) dynamically through the control loop.

3.1.6 Implementation of MSA based systems in DevOps

The identified problems and solutions in this category belong to microservices integration and managing databases for microservices. To deal with the operational and configuration complexity issue, Study (S20) recommends a CD platform, which provides a CD pipeline for each service that can give control over the integration of microservices. To address the complexity issues due to a large number of microservices, Study (S24) suggests two guidelines: first, keep the interface of each microservice as simple as possible for integration purposes, and second, it is recommended to use the technology which does not require specific programming language while implementing microservices to avoid from integration issues. Moreover, Study (S03) proposes a platform (i.e., HARNESS) to facilitate the integration of microservices that are developed in geographically distributed locations. In addition to these guidelines, Study (S08) also proposes the CIDE platform that provides precise control over testing, deployment, and integration of the new functionality into existing systems. To handle the problem of data management of MSA

based systems, Study (S24) discusses the use of database per service and a shared database for multiple microservices patterns. Database per service pattern can be implemented through defining a separate set of tables per function, scheme per service, and database server per service, whereas a shared database pattern can be implemented by defining a single database for a group of microservices. Usually, microservices are grouped according to the business context to use the shared database.

3.1.7 Testing of MSA based systems in DevOps

The number of services, inter-communication processes, dependencies, instances, and other variables influence the testing process for MSA based systems in DevOps. We identified six studies that stress on excessive testing of MSA based systems in DevOps. Study (S28) claims that all traditional testing strategies (e.g., unit testing, functional testing, regression testing, etc.) can be used to test MSA based systems. Moreover, Study (S28) also recommends internal testing, service testing, protocol testing, composition testing, protocol testing, scalability/throughput testing, failover/fault tolerance testing, and penetration testing strategies. Apart from the testing strategies mentioned above, Study (S08) and Study (S11) presents the CIDE platform that can be used to test MSA based systems in DevOps. The tools we identified from the selected studies that can be used to test MSA based systems are listed in Figure 7.

3.1.8 Deployment of MSA based systems in DevOps

Many solutions have been proposed to address the issues of MSA based system deployment in DevOps (e.g., complexity, dynamic deployment, and deployment in development, production, and testing environment). For instance, Study (S12) recommends a multipurpose Docker Compose tool, which can work in different environments, such as staging, development, deployment, and testing environments, and smooth the deployment process of microservices in the development environment. Study (S27) recommends Kubernetes, working with a range of containers tools (e.g., Dockers), to deploy and scale microservices into the production environment. Study (S20) recommends that the frequent deployment of microservices must be automated through a CD pipeline to finish within due time. To address the problem of complexity in dynamic deployment of many microservices, Study (S08) and Study (S11) present the CIDE platform, which provides precise control over dynamic deployment through Communication Engine (CE) and Local Execution Engine (LEE). To deal with the problem of MSA based SaaS deployment, Study (S21) proposes the SmartVM framework to automate the deployment of MSA based SaaS. Study (S21) also provides strategies (e.g., Traefik, HTTP reverse proxy, round-robin) for load balancing and separating the functional and operational concerns. Jolie Redeployment Optimiser (JRO) has been employed to achieve an optimal deployment of MSA based systems (S25). JRO consists of three components: Zephyrus, Jolie Enterprise (JE), and Jolie Reconfiguration Coordinator (JRE), in which Zephyrus generates detailed and optimal architecture for MSA based systems,

JE provides a framework for deploying and managing microservices, and JRE interacts with Zephyrus and JE for optimized deployment.

3.1.9 Monitoring of MSA based Systems in DevOps

A factory design pattern-based approach, called Omnia, has been proposed to address monitoring infrastructure problem (S05). This approach provides a component called monitoring interface, which enables developers to monitor MSA based systems independently and helps system administrators to build monitoring systems that are compatible with such interface by using monitoring factory components. Some tools can help address logging issues (see Figure 7). To address the problem of monitoring fine-grain microservices at runtime in a shared execution environment, Study (S18) presents DevOps based Unicorn framework, which can monitor highly decomposed MSA based systems at runtime (S18).

3.1.10 Organizational Problems

This theme reports problems related to culture, people, cost, and organization and team structure in the context of MSA and DevOps combination. To handle the problems that may be faced with when introducing MSA and DevOps combination in a given organization, Study (S23) suggests some guidelines, such as adopting new organizational structure, introducing small cross-functional teams, training for learning new skills (e.g., MSA, DevOps), changing employee habits toward the team work and sharing of responsibilities, and providing separate physical locations to teams, etc. Study (S24) suggests that the monolithic organizational structure needs to be aligned with the architecture of MSA based systems. Similarly, to address the issue related to establishing skilled and educated DevOps teams, Study (S24) suggests that the organization should arrange training programs for their employees for learning and adopting microservices in DevOps.

3.1.11 Resource Management Problems

This category provides the mapping of problems and solutions for different types of resources required to implement MSA in DevOps. Study (S01) recommends the virtualization of applications, infrastructures, and platforms resources as a solution for addressing resource management problems. Study (S09) suggests using containers and VMs for microservices in DevOps to get the desired level of efficiency in resource utilization. Study (S03) proposes the HARNESS approach (i.e., a DevOps based approach) that provides a cloud-based platform for bringing together commodity and specialized resources (e.g., skilled people). Study (S19) introduces an MSA based SONATA NFV platform with DevOps to address resource management problems by providing a set of tools (e.g., GitHub, Jenkins, Docker). The SONATA NFV platform can also create the CI/CD pipeline to automate steps in software delivery process. Study (S09) argued that dedicated access to the host's hardware can be increased either by giving extra

privileges to microservices or by enhancing the capability of containers to access the host resources.

3.2 Requisitos de sistemas baseados em microserviços em DevOps

- Performance Issue due to Lack of Dedicated Access to the Host's Hardware (S09) . Enable Dedicated Access to the Host's Hardware (S09)

- Empowering Developers through intelligent Software (S45) . Machine Learning-based Plug-In (S45)

- Performance Overhead due to Fine Grain Decomposition (S02, S06, S08, S11, S18, S24, S30, S33, S45) . Unicorn Framework (S18) . VM Auto-configuration Method (S18) . CIDE Platform (S08)

Scaling MSA-based Systems (S16, S18, S41) . Neo-Metropolis Pratfform (S16) . Designing microservices around business capabilites (S41) . Containers (S18)

3.3 Design de sistemas baseados em microserviços em DevOps

Security and Privacy Across Cloud-Native Applications (S10, S18, S20, S23) . ARCADIA framework (S10)

3.4 Do monolito aos microserviços

Como migrar do monolito para os microserviços

Plano de Continuidade

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 2 vezes nas páginas 10 e 13.

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 14.

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 na página 15.

Apêndices

APÊNDICE A – Quisque libero justo

Quisque facilisis auctor sapien. Pellentesque gravida hendrerit lectus. Mauris rutrum sodales sapien. Fusce hendrerit sem vel lorem. Integer pellentesque massa vel augue. Integer elit tortor, feugiat quis, sagittis et, ornare non, lacus. Vestibulum posuere pellentesque eros. Quisque venenatis ipsum dictum nulla. Aliquam quis quam non metus eleifend interdum. Nam eget sapien ac mauris malesuada adipiscing. Etiam eleifend neque sed quam. Nulla facilisi. Proin a ligula. Sed id dui eu nibh egestas tincidunt. Suspendisse arcu.

APÊNDICE B – Nullam elementum urna vel imperdiet sodales elit ipsum pharetra ligula ac pretium ante justo a nulla curabitur tristique arcu eu metus

Nunc velit. Nullam elit sapien, eleifend eu, commodo nec, semper sit amet, elit. Nulla lectus risus, condimentum ut, laoreet eget, viverra nec, odio. Proin lobortis. Curabitur dictum arcu vel wisi. Cras id nulla venenatis tortor congue ultrices. Pellentesque eget pede. Sed eleifend sagittis elit. Nam sed tellus sit amet lectus ullamcorper tristique. Mauris enim sem, tristique eu, accumsan at, scelerisque vulputate, neque. Quisque lacus. Donec et ipsum sit amet elit nonummy aliquet. Sed viverra nisl at sem. Nam diam. Mauris ut dolor. Curabitur ornare tortor cursus velit.

Morbi tincidunt posuere arcu. Cras venenatis est vitae dolor. Vivamus scelerisque semper mi. Donec ipsum arcu, consequat scelerisque, viverra id, dictum at, metus. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut pede sem, tempus ut, porttitor bibendum, molestie eu, elit. Suspendisse potenti. Sed id lectus sit amet purus faucibus vehicula. Praesent sed sem non dui pharetra interdum. Nam viverra ultrices magna.

Aenean laoreet aliquam orci. Nunc interdum elementum urna. Quisque erat. Nullam tempor neque. Maecenas velit nibh, scelerisque a, consequat ut, viverra in, enim. Duis magna. Donec odio neque, tristique et, tincidunt eu, rhoncus ac, nunc. Mauris malesuada malesuada elit. Etiam lacus mauris, pretium vel, blandit in, ultricies id, libero. Phasellus bibendum erat ut diam. In congue imperdiet lectus.

Anexos

ANEXO A – Morbi ultrices rutrum lorem.

Sed mattis, erat sit amet gravida malesuada, elit augue egestas diam, tempus scelerisque nunc nisl vitae libero. Sed consequat feugiat massa. Nunc porta, eros in eleifend varius, erat leo rutrum dui, non convallis lectus orci ut nibh. Sed lorem massa, nonummy quis, egestas id, condimentum at, nisl. Maecenas at nibh. Aliquam et augue at nunc pellentesque ullamcorper. Duis nisl nibh, laoreet suscipit, convallis ut, rutrum id, enim. Phasellus odio. Nulla nulla elit, molestie non, scelerisque at, vestibulum eu, nulla. Ut odio nisl, facilisis id, mollis et, scelerisque nec, enim. Aenean sem leo, pellentesque sit amet, scelerisque sit amet, vehicula pellentesque, sapien.

ANEXO B – Cras non urna sed feugiat cum sociis natoque penatibus et magnis dis parturient montes nascetur ridiculus mus

Sed consequat tellus et tortor. Ut tempor laoreet quam. Nullam id wisi a libero tristique semper. Nullam nisl massa, rutrum ut, egestas semper, mollis id, leo. Nulla ac massa eu risus blandit mattis. Mauris ut nunc. In hac habitasse platea dictumst. Aliquam eget tortor. Quisque dapibus pede in erat. Nunc enim. In dui nulla, commodo at, consectetuer nec, malesuada nec, elit. Aliquam ornare tellus eu urna. Sed nec metus. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas.

ANEXO C – Fusce facilisis lacinia dui

Phasellus id magna. Duis malesuada interdum arcu. Integer metus. Morbi pulvinar pellentesque mi. Suspendisse sed est eu magna molestie egestas. Quisque mi lorem, pulvinar eget, egestas quis, luctus at, ante. Proin auctor vehicula purus. Fusce ac nisl aliquam ante hendrerit pellentesque. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Morbi wisi. Etiam arcu mauris, facilisis sed, eleifend non, nonummy ut, pede. Cras ut lacus tempor metus mollis placerat. Vivamus eu tortor vel metus interdum malesuada.