Padrões de migração de sistemas legados para arquitetura baseada em microsserviços

Migration patterns of legacy systems to microservices based architecture

Rafael de Carvalho Cintra[1], Wilson Vendramel[1]

[1]Instituto Federal de Educação, Ciência e Tecnologia de São Paulo (IFSP)
Bragança Paulista – SP – Brasil

rafaelcintra@gmail.com, wvendramel@ifsp.edu.br

Abstract. *Microservice is a software architectural style that has been consolidating in application development projects, consisting in a set of small services. In the same direction, there's a big demand to migrate legacy systems built in monolithic architecture to this new style with the purpose to take advantage of the technologies from distributed systems. This paper aims to list patterns that can be used in projects that target to migrate an application developed using monolithic approach to a new microservices approach. A survey with IT professionals was executed to validate the applicability of these patterns. Finally, analyzing the results, it observes that the migration patterns listed are treated with relevancy by the researched professionals.*

Keywords: *Software Architecture. Monolithic Architecture. Microservices. Migration Patterns. Legacy Systems.*

Resumo. *A arquitetura de microsserviços é um estilo de arquitetura de software que vem se consolidando em projetos de desenvolvimento de aplicações, consistindo em um conjunto de pequenos serviços. Da mesma forma, existe uma grande demanda para a migração de sistemas legados de arquiteturas monolíticas para essa nova arquitetura, a fim de que essas soluções se beneficiem das novas tecnologias de sistemas distribuídos. Este trabalho tem como objetivo listar padrões que possam ser adotados em projetos de migração de uma aplicação desenvolvida sobre a arquitetura monolítica para uma arquitetura de microsserviços. Uma pesquisa com profissionais de TI foi realizada para validar a aplicabilidade desses padrões. Por fim, analisando os resultados, observa-se que os padrões de migração listados são tratados com relevância pelos profissionais pesquisados.*

Palavras-chave: *Arquitetura de Software. Arquitetura Monolítica. Microsserviços. Padrões de Migração. Sistemas Legados.*

1. Introdução

Microsserviços é um estilo de arquitetura de software baseado na arquitetura em nuvem onde um sistema de software é desenvolvido como um conjunto de pequenos serviços (KRATZKE e QUINT, 2017). Cada um desses serviços é capaz de ser instalado de forma independente e em diferentes plataformas, rodando o seu próprio processo e se comunicando entre si por meio de mecanismos leves tais como RESTful APIs. Nesse contexto, cada serviço executa uma função de negócio e cada um deles pode ser implementado utilizando diferentes linguagens de programação e armazenamento de dados (FOWLER e LEWIS, 2014).

A migração de um sistema para arquitetura de microsserviços trás diversos benefícios, dentre elas a possibilidade de prover disponibilidade e escalabilidade considerando as diferentes partes do sistema, ou seja, prover diferentes estruturas de hardware para distintas partes do sistema de forma independente e de acordo com a sua necessidade. Outro beneficio é a possibilidade da utilização de diferentes tecnologias para os diferentes serviços, evitando assim que o sistema fique amarrado a apenas uma tecnologia e seus benefícios (NEWMAN, 2015).

O sistema migrado pode fazer uso da elasticidade provida pelos serviços em nuvem, que têm a capacidade de se ajustarem à necessidade da aplicação, oferecendo uma melhor experiência ao usuário, além de diminuir o custo de operação e manutenção de servidores (BALALAIE, HEYDARNOORI e JAMSHIDI, 2015).

Apesar da arquitetura de microsserviços apresentar muitos benefícios, ela também traz complexidade ao sistema onde novos componentes são necessários, tais como registro de serviços. A decomposição do sistema em pequenos serviços, o monitoramento e gerenciamento desses serviços entre outros fatores importantes, também precisam ser considerados (BALALAIE, HEYDARNOORI e JAMSHIDI, 2016).

Em um projeto sem uma metodologia bem definida, a migração pode se tornar uma sequência de tentativas e erros, resultando não somente em uma grande perda de tempo, mas também em uma solução mal desenvolvida (JAMSHIDI, AHMAD e PAHL, 2013). Vale ressaltar que considerando a variedade de fatores tais como requisitos, conhecimento dos membros da equipe e cenários encontrados nas organizações, uma única e rígida metodologia pode não ser adequada.

Muitas empresas ainda hesitam em migrar aplicações sobre a arquitetura monolítica para arquitetura de microsserviços porque ainda desconhecem o processo de migração e as questões e benefícios relacionados à essa migração (TAIBI, LENARDUZZI e PAHL, 2017).

A arquitetura monolítica é um estilo arquitetural em que a lógica da aplicação é instalada em uma única unidade. Isso não significa que esse estilo seja ruim em todos os casos, já que a arquitetura monolítica pode ser a mais apropriada para sistemas pequenos, pois um simples balanceamento de carga pode garantir alta escalabilidade e disponibilidade (BALALAIE, HEYDARNOORI e JAMSHIDI, 2015).

Dentre os trabalhos relacionados, vale destacar o estudo de Taibi, Lenarduzzi e Pahl (2017) que realizou uma pesquisa com profissionais experientes que já migraram monólitos para microsserviços, comparando três processos de migração distintos adotados pelos profissionais entrevistados, juntamente com as questões e motivações comuns que costumam acontecer durante as migrações. A manutenibilidade e a escalabilidade foram considerados como as motivações mais importantes ao se decidir pela migração.

O objetivo deste trabalho é identificar na literatura padrões de migração que tragam benefícios e facilidades em projetos que visam migrar uma aplicação desenvolvida sobre a arquitetura monolítica para uma arquitetura de microsserviços. Para tal, foram realizadas pesquisas bibliográficas que trouxeram embasamento teórico a respeito dos padrões elencados. Além do mais, foi realizada uma pesquisa com profissionais da área de Tecnologia da Informação com a finalidade de avaliar a aplicabilidade e o grau de relevância dos padrões na prática do dia a dia.

Além desta seção de introdução, este trabalho fundamenta conceitos sobre os padrões de migração relacionados para este estudo na seção 2, a aplicação da pesquisa na seção 3, a análise dos resultados na seção 4 e, por fim, a seção 5 contendo as conclusões.

2. Padrões de Migração

A migração da arquitetura de um sistema monolítico para arquitetura de microsserviços trás diversos benefícios como a possibilidade de controlar a disponibilidade e escalabilidade dos diferentes componentes da aplicação, a possibilidade de utilizar diferentes tecnologias em diferentes pontos da solução, podendo assim aplicar a melhor tecnologia disponível para os diferentes problemas endereçados nos diversos componentes (BALALAIE, HEYDARNOORI e JAMSHIDI, 2016).

Apesar da arquitetura de microsserviços trazer muitos benefícios, ela também traz complexidades ao sistema o qual precisa adotar novos componentes. A dificuldade na decomposição de grandes serviços em serviços menores, a necessidade de monitoramento e gerenciamento dos serviços são alguns dos fatores que tornam a migração para microsserviços uma tarefa desafiadora. Desafios como esses são minimizados com a adoção de metodologias, práticas, artefatos e padrões de design descritos a seguir os quais são denominados neste trabalho como padrões de migração.

2.1 Implementar e suportar integração continua

Considerando o aumento do número de serviços que a arquitetura de microsserviços trás e a complexidade no gerenciamento do sistema em ambiente de produção, é necessário um mecanismo que ajude a manter esse ambiente sempre preparado para entrega contínua, ou seja, a atualização e instalação de novas funcionalidades.

O primeiro passo para garantir entrega contínua é a adoção de um processo de integração contínua. Entrega contínua é uma prática DevOps que provê a possibilidade de entrega de uma nova versão de um serviço em qualquer ambiente de forma automatizada. O processo de integração contínua permite que os desenvolvedores

integrem seu desenvolvimento com o de outros desenvolvedores de forma regular ajudando a resolver futuros conflitos (HUMBLE e FARLEY, 2010). A integração contínua também é responsável por automatizar os processos de *build* e teste para garantir a disponibilidade dos artefatos em produção. Nesse processo, basicamente cada serviço deve ter seu próprio repositório para manter seu próprio histórico de alterações e que seja possível fazer um processo de *build* independente para cada um deles. Toda vez que o código no repositório for alterado, um processo automático deve ser iniciado, sendo que este deve gerar um artefato a partir da última versão disponível, rodar todos os testes configurados e fazer a entrega no repositório de artefatos. Qualquer erro ou falha nesse processo deve ser informado aos desenvolvedores. Uma regra simples da integração contínua é que novas alterações não devem quebrar ou prejudicar a estabilidade do sistema e deve passar em todos os testes pré-definidos.

2.2 Decomposição do sistema monolítico

Alguns problemas são comuns de serem encontrados em aplicações monolíticas. A complexidade de um código que se torna difícil de ser compreendido, as diferentes partes do sistema que possuem requisitos não funcionais desnecessários e a necessidade de uma instalação total da aplicação mesmo quando a alteração se refere a apenas uma pequena parte dela são apenas alguns exemplos que podem se resolver utilizando microsserviços. Para isso há a opção de redesenhar a solução monolítica para um conjunto de serviços com base na metodologia de Domain-Driven Design (DDD). Vale ressaltar que é recomendado começar com um número pequeno de serviços, dois ou três, e aumentar de acordo com o aumento de conhecimento do negócio (EVANS, 2004).

DDD também pode ser visto como uma prática comum para transformar a arquitetura do sistema em microsserviços (VERNON, 2013), sendo uma ótima opção para a decomposição inicial do sistema para identificar subdomínios do negócio o qual o sistema opera, sendo que o subdomínio identificado vai formar um escopo de negócios que será uma unidade de *deploy*. Essa unidade também pode ser decomposta com base nos requisitos não funcionais ou na frequência que as diferentes partes da unidade são atualizadas, para tal também recomenda-se utilizar a metodologia DDD. É difícil sugerir o tamanho das unidades de *deploy*, pois é totalmente dependente dos requisitos que podem mudar, isto é, os requisitos podem aumentar ou diminuir com o passar do tempo.

2.3 Minimizar a dependência de código utilizando chamadas de serviços

É uma boa prática manter os serviços o máximo possível separados. A alteração de um código que é compartilhado em várias partes do sistema aumenta a possibilidade de quebrar o processo de *build*. Entretanto, isso não costuma se aplicar às bibliotecas compartilhadas que raramente se alteram, como por exemplo as bibliotecas nativas de linguagem para manipulação de String, sendo recomendado o compartilhamento de código. Para bibliotecas internas desenvolvidas para a solução, essa dependência deve ser evitada, pois qualquer alteração na mesma implica em mudança no código que possui sua dependência (BALALAIE et al., 2018).

Nos casos em que compartilhar a biblioteca não é uma boa opção, sugere-se compartilhar a funcionalidade como um serviço, que pode ser um serviço totalmente

isolado ou parte de um outro serviço. Como consequência dessa alteração, há a possibilidade de se oferecer uma escalabilidade diferente para o serviço compartilhado e aos serviços que o utilizam. Possivelmente isso será necessário porque o serviço compartilhado será requerido por diversos outros, necessitando maior desempenho.

Em alguns casos, a utilização de uma biblioteca utilizando chamada e serviço ao invés de chamada direta de biblioteca pode causar problemas de performance, isso deve ser analisado e ponderado ao selecionar a biblioteca que será migrada. A adoção de mecanismos de *cache* para o serviço compartilhado também pode ser estudado e implementado a fim de diminuir esse impacto, conforme mostra a Figura 1 (BALALAIE et al., 2018).

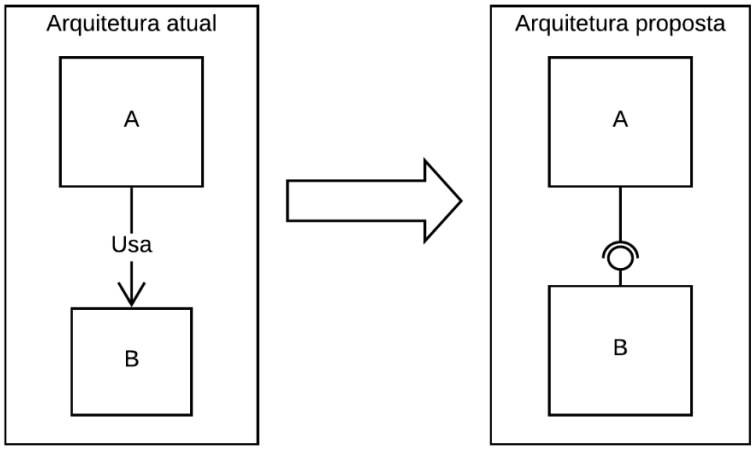

Figura 1. Substituir a dependência de código por chamadas de serviços
Fonte: adaptado de Balalaie et al (2018)

2.4 Aplicar o padrão Strangler Application

O padrão Strangler Application tem por objetivo criar uma camada ou aplicação que intercepte as chamadas feitas para a aplicação a ser migrada e redireciona aos novos microsserviços, quando os mesmos já estiverem implementados (FOWLER e LEWIS, 2014).

Implementar e migrar toda a aplicação legada para a arquitetura de microsserviços de uma vez é uma tarefa um tanto quanto arriscada, portanto a aplicação desse padrão reduz os riscos da migração, permitindo que os serviços sejam migrados gradualmente. Como se pode observar na Figura 2, o padrão Strangler Application não elimina a aplicação legada, mas integra os novos serviços aos legados.

A aplicação do Strangler Application introduz dois componentes:

a) Dispatcher: esse componente funciona como um roteador, gerenciando as requisições e enviando aos novos serviços ou para a antiga aplicação monolítica;

b) IC: esse componente é implementado nos novos microsserviços e na aplicação monolítica, e são responsáveis pela integração entre a aplicação legada e os novos serviços (SANTIS et al., 2016).

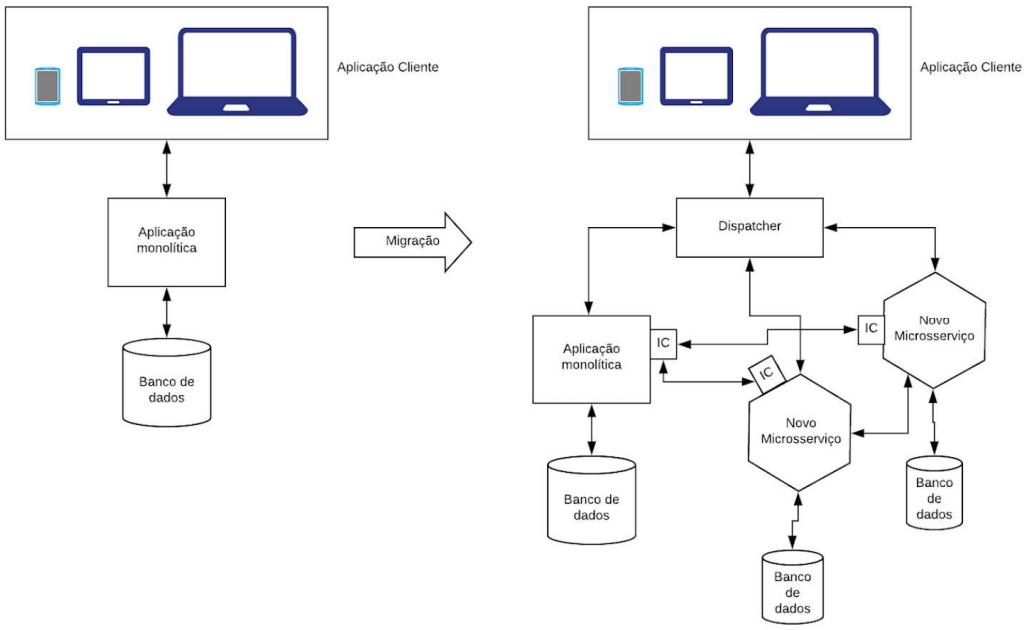

Figura 2. Strangler Application
Fonte: Adaptado de Santis et al (2016)

2.5 Criar um registro de serviços

Na arquitetura de microsserviços, o conjunto das instâncias dos serviços se alteram dinamicamente. Consequentemente, para que a aplicação cliente possa fazer uma requisição a um serviço, é preciso acessar um mecanismo em que possa consultar a disponibilidade e acessibilidade ao mesmo. Para tal funcionalidade, utiliza-se o registro de serviços, que consiste em uma aplicação que gerencia o banco de dados de serviços. Esse padrão disponibiliza uma API (Application Programming Interface) para adicionar e outra para remover o registro do serviço, conforme mostra a Figura 3, assim a aplicação que o consumirá terá acesso a sua disponibilidade e localidade (BAKSHI, 2017). Outra forma de manter o banco de dados atualizado é implementar no serviço de registro uma lógica que consulte a disponibilidade do serviço periodicamente para garantir sua disponibilidade, enviando um *heartbeat* (BALALAIE et al.,2018).

Para consultar a disponibilidade do serviço, o cliente pode utilizar dois mecanismos: *client-side discovery* e *server-side discovery.* No mecanismo de *client-side discovery*, a aplicação consumidora consulta o registro de serviços, seleciona uma instância disponível e executa a chamada. Enquanto que no mecanismo *server-side discovery,* a aplicação consumidora envia a requisição para a aplicação de registro de serviços, esta seleciona uma instância disponível do recurso a ser acessado e encaminha a chamada, conforme exibe a Figura 4 (BAKSHI, 2017).

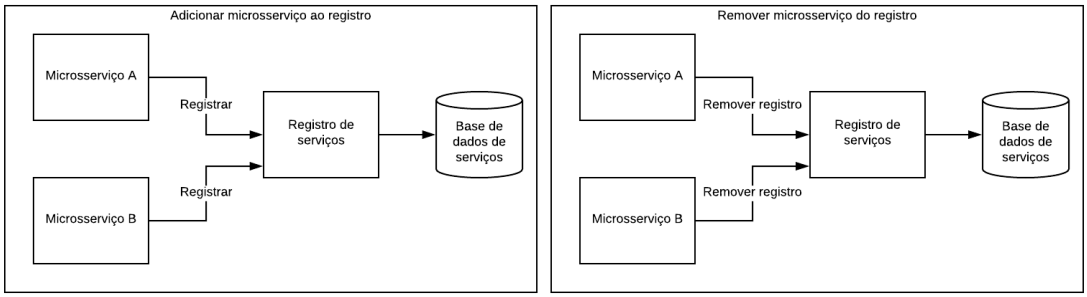

Figura 3. Registro de serviços
Fonte: Adaptado de Bakshi (2017)

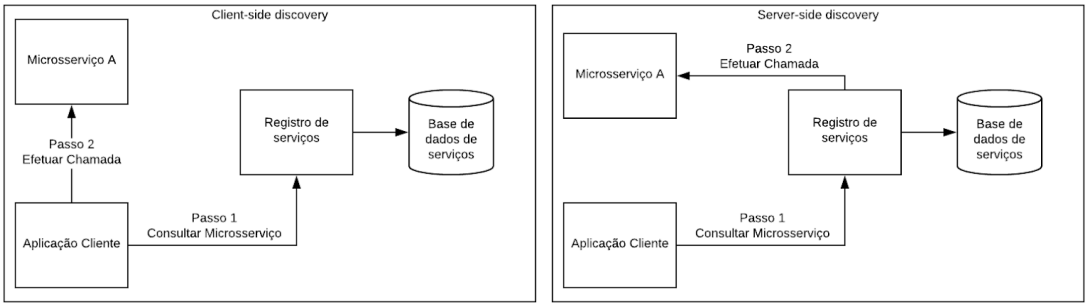

Figura 4. Mecanismos de client-side discovery e server-side discovery
Fonte: Adaptado de Bakshi (2017)

2.6 Criar um Load Balancer

Com a migração, decomposição e desenvolvimento de diversos pequenos serviços utilizando arquitetura de microsserviços, uma ou mais instâncias do mesmo serviço poderão e deverão estar disponíveis para os clientes. O Load Balancer é um componente arquitetural responsável por distribuir as chamadas para as diversas instâncias de serviços utilizando uma lógica própria a fim de garantir a escalabilidade dos serviços, conforme exibe a Figura 5 (BALALAIE, HEYDARNOORI e JAMSHIDI, 2016). O Load Balancer poderá manter sua lista de serviços atualizada, utilizando o padrão de registro de serviços já citado na seção anterior. Sua lógica de distribuição das requisições para os microsserviços pode obedecer a uma série de critérios específicos, entre eles:

a) Distribuição assimétrica: requisições mais pesadas devem ser encaminhadas para serviços com maior disponibilidade de processamento;

b) Requisições por prioridade: as requisições são gerenciadas de acordo com o nível de prioridade atribuído a ela;

c) Distribuição pelo conteúdo: as requisições são distribuídas aos recursos de acordo com o conteúdo da requisição.

O Load Balancer está geralmente localizado no nível de comunicação entre o cliente que utiliza o serviço e o serviço oferecido, e pode ser projetado de forma

transparente, tanto para o lado dos microsserviços quanto para o lado do cliente (CLOUD COMPUTING PATTERNS, 2018).

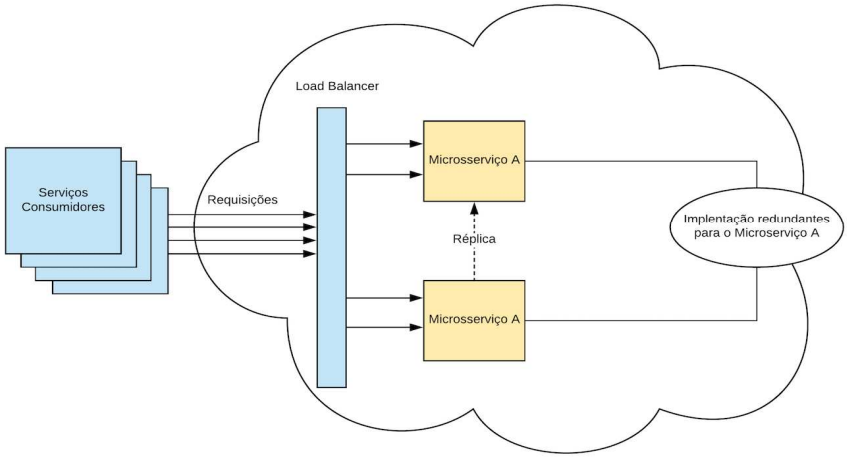

Figura 5. Load Balancer
Fonte: Adaptado de Cloud Computing Patterns (2018)

2.7 Criar um servidor de configuração

Após o sistema ter sido decomposto da arquitetura monolítica para a arquitetura de microsserviços, haverá um conjunto de pequenos serviços os quais possuem suas próprias configurações. Em um mundo ideal, essas configurações deveriam ser limitadas às diferenças entre os ambientes de desenvolvimento, teste e produção a fim de diminuir as chances de encontrar problemas quando o serviço migrar de um ambiente para outro (NEWMAN, 2015).

Uma boa forma de gerenciar as configurações é mantê-las em um artefato separado da aplicação, para tal cria-se um servidor de configurações. Esse é um serviço que provê aos microsserviços as configurações necessárias para que executem suas tarefas. Além disso, qualquer alteração de alguma propriedade será propagada automaticamente para os serviços que a utilizam sem a necessidade de uma nova instalação do serviço (BALALAIE et al.,2018). A Figura 6 representa a interação entre os microsserviços e o servidor de configuração.

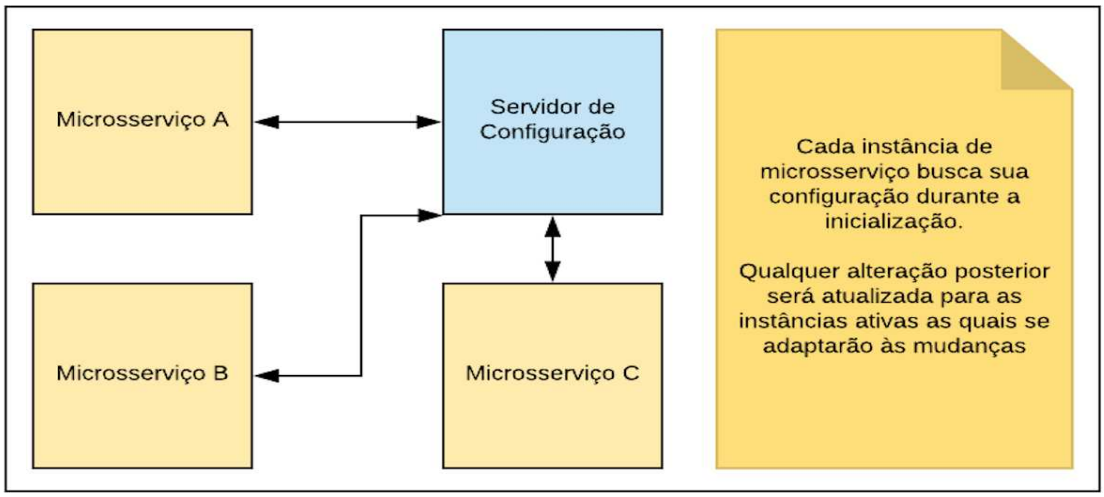

Figura 6. Servidor de Configuração
Fonte: Adaptado de Balalaie et al (2018)

3. Aplicação da Pesquisa

Para validação dos padrões relacionados neste trabalho, foi realizada uma pesquisa via Web com 40 profissionais da área de Tecnologia da Informação durante o mês de novembro de 2018, especificamente das cidades de Bragança Paulista, Campinas e São Paulo, com a finalidade de avaliar o grau de importância dos padrões de migração apresentados.

O instrumento de pesquisa adotado foi o questionário do tipo *survey*, contendo nove questões, das quais duas delas buscam extrair informações sobre a área de atuação e nível de senioridade dos pesquisados, enquanto as outras sete questões estão relacionadas diretamente com os padrões de migração. Durante o estudo foram enviadas mensagens diretas e e-mails aos profissionais que participaram da pesquisa.

3.1 Perfil dos pesquisados

Na primeira questão **"Qual a sua área de atuação na atual empresa em que trabalha?"**, 60% dos profissionais pesquisados atuam na área de desenvolvimento e manutenção de software, 22,5% na área de arquitetura de software, 7,5% na área de gerência de projetos e 10% em outras áreas de Tecnologia da Informação. Nenhum profissional pesquisado atua na área de infraestrutura e redes, como mostra a Figura 7.

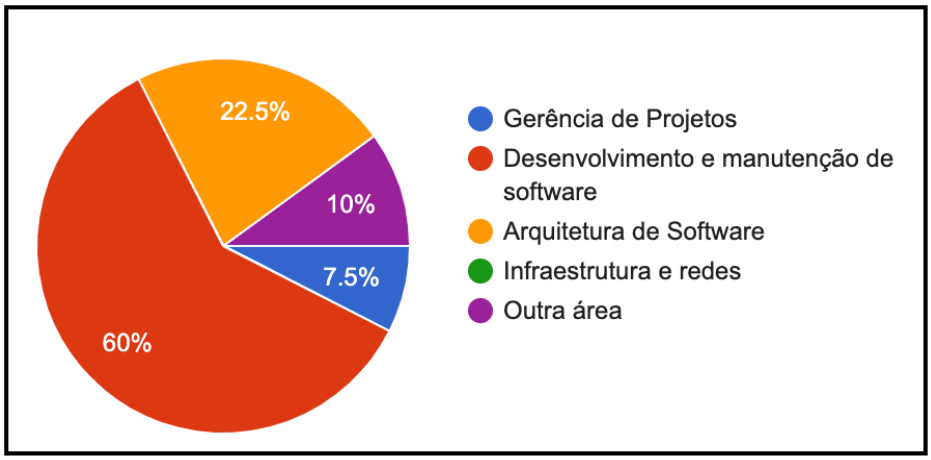

Figura 7. Área de atuação

Na segunda questão **"Há quanto tempo você atua na área de TI?"** que busca coletar informação referente a senioridade dos profissionais, 52,5% dos profissionais pesquisados atuam na área de Tecnologia da Informação há mais de 10 anos, 20% entre 5 e 10 anos, 15% entre 2 e 5 anos e, finalmente, 12,5% menos de 2 anos, como apresenta a Figura 8.

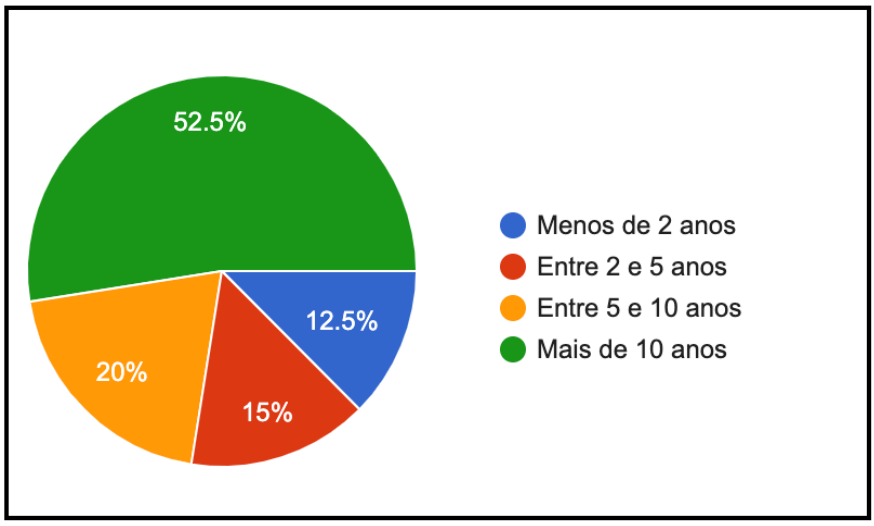

Figura 8. Tempo de atuação na área de TI

3.2 Integração Contínua

Referente à afirmação de número 3 **"O processo de integração contínua é de extrema importância para um projeto de migração de arquitetura monolítica para arquitetura de microsserviços"**, 77,5% dos pesquisados responderam que concordam plenamente com a afirmação, 15% concordam parcialmente e 7,5% não conhecem ou não souberam responder, conforme mostra a Figura 9. O resultado mostra que a

integração contínua é considerada importante pela maioria dos profissionais pesquisados, dando validade a esse padrão em projetos de migração.

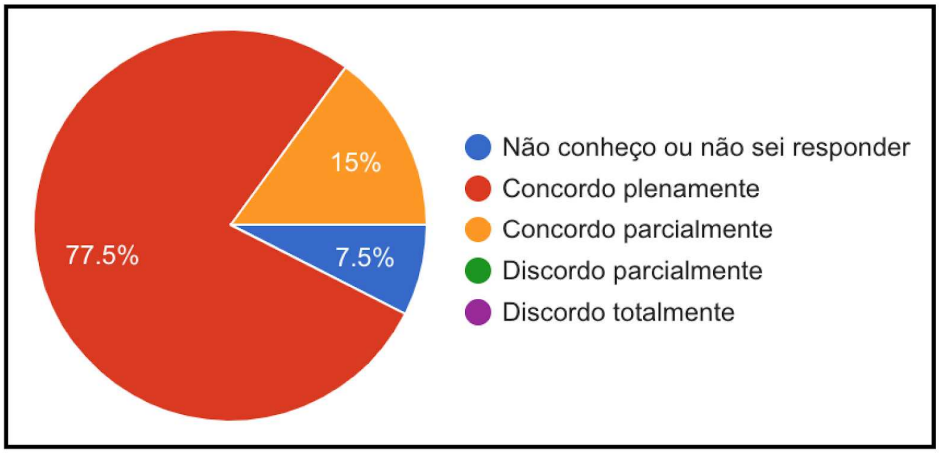

Figura 9. Integração Contínua

3.3 Decomposição utilizando DDD (Domain-Driven Design)

Quando apresentada a afirmação de número 4 **"A metodologia Domain-Driven Design (DDD) é de extrema importância para decompor o sistema monolítico em pequenos serviços"**, 45% dos pesquisados responderam que concordam parcialmente, 32,5% concordam plenamente, 7,5% discordam parcialmente e 15% não conhecem ou não souberam responder, conforme exibe a Figura 10. Grande parte dos profissionais pesquisados considera a metodologia DDD como uma prática importante para auxiliar na decomposição do sistema em pequenos serviços, visando projetos de migração.

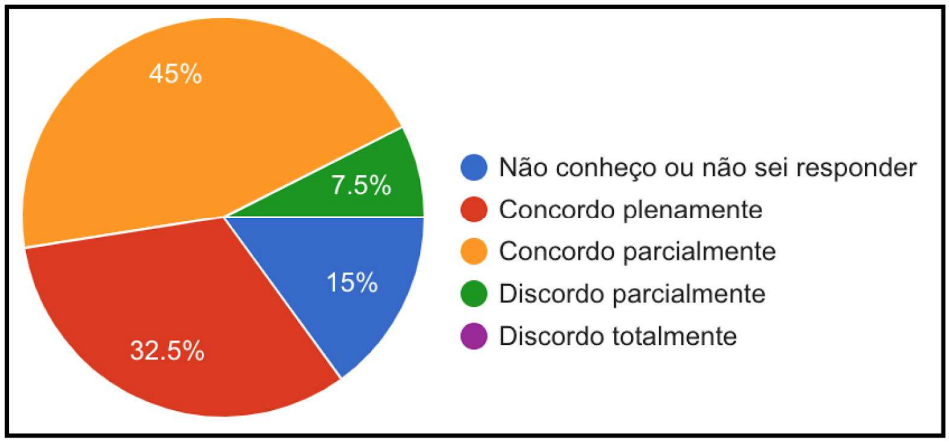

Figura 10. Decomposição utilizando DDD

3.4 Minimizar dependência de código utilizando chamadas de serviços

Referente à afirmação de número 5 **"É uma boa prática em uma arquitetura de microsserviços manter os serviços o máximo possível separados, para isso sugere-se a substituição da dependência de código pela utilização de chamada de serviço"**, 52,5% dos pesquisados responderam que concordam plenamente com a afirmação, 35% concordam parcialmente e 12,5% não conhecem ou não souberam responder, conforme apresenta a Figura 11. A pesquisa mostrou uma grande aceitação por parte dos profissionais pesquisados, trazendo a ideia de que a diminuição de dependência de código utilizando chamadas de serviços é muito adotada em projetos de migração. Vale ressaltar que nenhum participante da pesquisa discordou dessa afirmação.

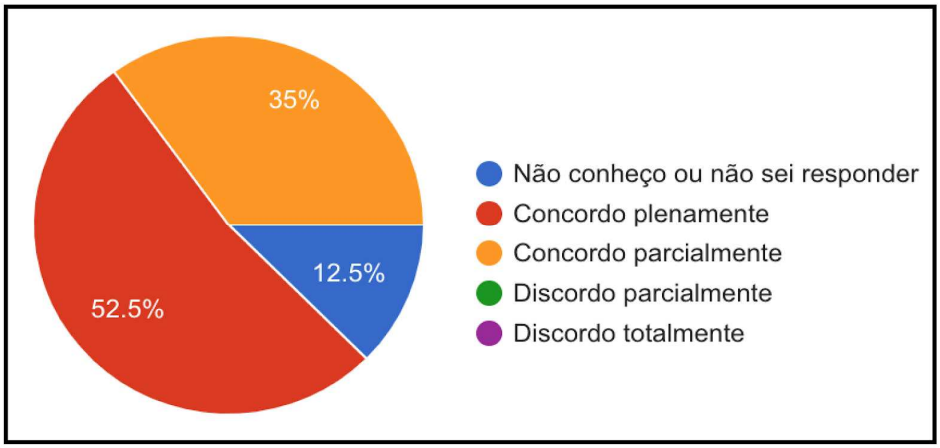

Figura 11. Minimizar dependência de código por chamadas de serviços

3.5 Aplicação do padrão Strangler Application

Quando apresentada a afirmação de número 6 **"O padrão strangler application é de grande importância em um projeto de migração de arquitetura monolítica para arquitetura de microsserviços"**, 42,5% dos pesquisados responderam que concordam plenamente, 17,5% concordam parcialmente, 5% discordam parcialmente e 35% não conhecem ou não souberam responder, conforme mostra a Figura 12. Identifica-se um grande percentual de profissionais que não conhecem ou não souberam responder, demonstrando que esse padrão ainda não é muito difundido. Entretanto, o grande número de pesquisados que concordam com a afirmação nos leva ao entendimento de que o padrão é de grande importância na migração de sistemas legados de arquitetura monolítica para arquitetura de microsserviços. Quanto mais esse padrão for divulgado e explicado, maior a tendência de ser adotado em projetos de migração.

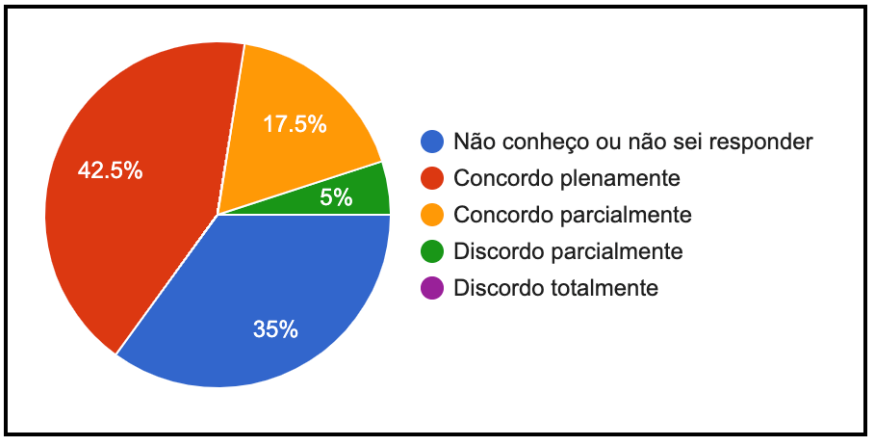

Figura 12. Aplicação do padrão Strangler Application

3.6 Criar um Registro de Serviços

Referente à afirmação de número 7 **"O registro de serviços é de grande importância em um projeto de migração de arquitetura monolítica para arquitetura de microsserviços"**, 50% dos pesquisados concordam plenamente, 30% concordam parcialmente, 5% discordam parcialmente e 15% não conhecem ou não souberam responder, conforme apresenta a Figura 13. O registro de serviços foi um padrão considerado bastante importante para projetos de migração de aplicações legadas de arquitetura monolítica para arquitetura de microsserviços, mostrando um baixo nível de discordância.

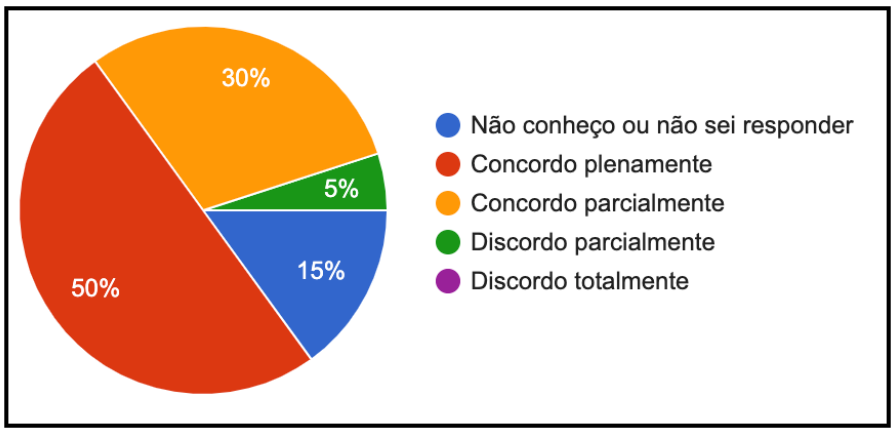

Figura 13. Registro de Serviços

3.7 Criar um Load Balancer

Quando apresentada a afirmação de número 8 **"É de grande relevância a adoção de um load balancer em um projeto de migração de arquitetura monolítica para arquitetura de microsserviços"**, 62,5% dos pesquisados responderam que concordam plenamente, 20% concordam parcialmente, 5% discordam parcialmente, 2,5% discordam totalmente e 10% não conhecem ou não souberam responder, conforme exibe

a Figura 14. O padrão Load Balancer apresenta um elevado nível de concordância plena, demonstrando ser poderoso e útil em projetos de migração.

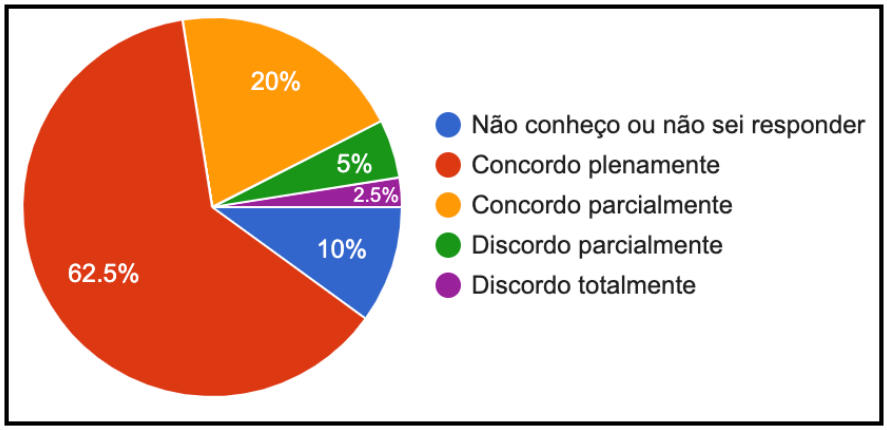

Figura 14. Load Balancer

3.8 Criar um servidor de configuração

Referente à afirmação de número 9 **"O servidor de configuração é de extrema importância em um projeto de migração de arquitetura monolítica para arquitetura de microsserviços"**, 45% dos pesquisados responderam que concordam plenamente, 27,5% concordam parcialmente, 12,5% discordam parcialmente, 5% discordam totalmente e 10% não conhecem ou não souberam responder, conforme mostra a Figura 15. Dentre todos os padrões apresentados, o servidor de configuração foi o padrão mais discordado, mas mesmo assim conta com em elevado nível de concordância entre os participantes da pesquisa. Assim como os outros, esse padrão se mostra de grande utilidade em projetos de migração de sistemas legados de arquitetura monolítica para arquitetura de microsserviços.

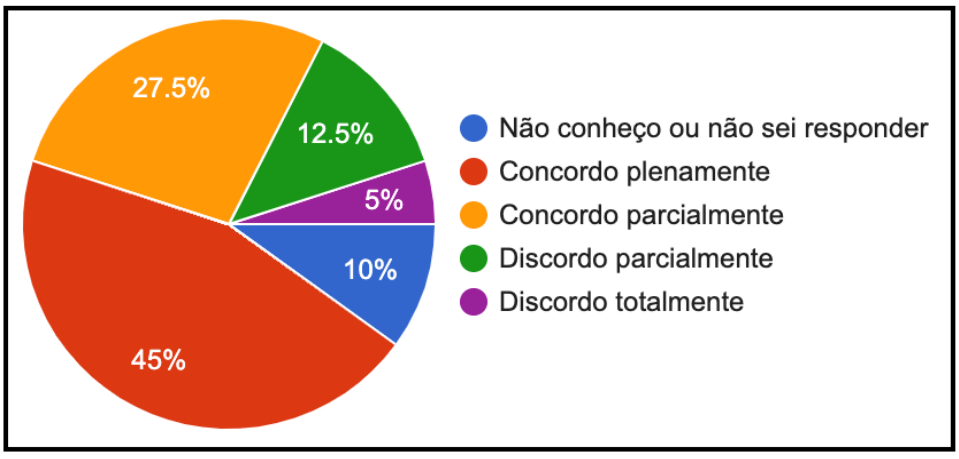

Figura 15. Servidor de Configuração

4. Análise dos Resultados

Para a análise dos resultados, o padrão será considerado de grande importância quando a soma das afirmações “Concordo plenamente” e “Concordo parcialmente” superar as somas dos valores das afirmações “Discordo parcialmente” e “Discordo totalmente”. Os resultados obtidos podem ser visualizados na Figura 16.

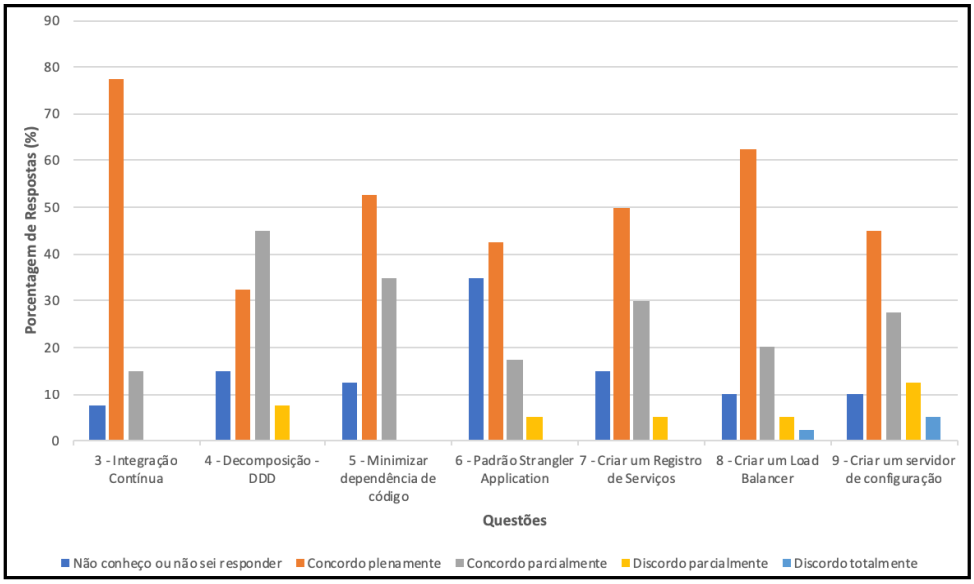

Figura 16. Resultados obtidos

A análise mostra que todos os padrões listados são de grande importância para um projeto de migração de arquitetura monolítica para arquitetura de microsserviços. Todas as somas de afirmações “Concordo plenamente” e “Concordo parcialmente” foram acima de 60%. Vale destacar que o padrão de Integração Contínua tem a maior relevância segundo os participantes desta pesquisa, totalizando 92,5% dos pesquisados. O segundo padrão de maior importância segundo os pesquisados foi a Minimização da dependência de código, os quais 87,5% dos pesquisados concordam que a dependência de código deve ser minimizada e substituída por chamadas de serviços. O terceiro padrão mais importante segundo os pesquisados é a criação de um Load Balancer para balancear a carga dos servidores com 82,5%.

Outra análise sobre os resultados obtidos permite constatar que os padrões mais desconhecidos pelos participantes da pesquisa são o padrão Strangler Application, os quais 35% dos pesquisados afirmam não conhecer ou não saber responder, sendo seguido pelos padrões Decomposição utilizando DDD e a Criação de um registro de serviços, ambos com 15%.

5. Conclusões

Baseado nos resultados apresentados, observa-se que os padrões de migração listados pela pesquisa bibliográfica neste estudo são conhecidos pela maioria dos profissionais pesquisados. Os padrões são considerados relevantes em projetos de migração de uma

aplicação legada baseada na arquitetura monolítica para uma nova aplicação desenvolvida utilizando a arquitetura de microsserviços. Vale destacar o nível de experiência dos profissionais envolvidos na pesquisa, sendo que 52,5% dos profissionais têm mais de 10 anos de experiência e apenas 12,5% têm menos de dois anos de atuação na área de TI, indicando que as respostas trazem um nível de confiança condizente, dado o nível de experiência dos participantes. Também se destaca que o nível de adesão à pesquisa pelos profissionais que atuam diretamente no desenvolvimento e arquitetura de software foi alto, contando com 82,5% da amostra total, trazendo benefícios à pesquisa, já que esses profissionais trabalham diretamente ligados ao tema de pesquisa abordado neste trabalho.

Por fim, mas não menos importante, nota-se que a adoção dos padrões de migração na concepção de um projeto que visa migrar um sistema legado de uma arquitetura monolítica para uma arquitetura de microsserviços ajudam não só a mitigar os riscos, mas também a organizar o projeto como um todo, executando a transição de maneira gradual e segura.

Referências

Bakshi, K. Microservices-based software architecture and approaches. 2017 IEEE Aerospace Conference, 2017, Big Sky, MT, USA.

Balalaie A.; Heydarnoori A.; Jamshidi P. Migrating to cloud-native architectures using microservices: an experience report. Advances in Service-Oriented and Cloud Computing: Workshops of ESOCC 2015, 2015, Taormina, Italy.

Balalaie A.; Heydarnoori A.; Jamshidi P. Microservices architecture enables DevOps: migration to a cloud-native architecture. IEEE Software, vol. 33(3), 2016, pp. 42-52.

Balalaie A.; Heydarnoori A.; Jamshidi P; Tamburri D. A.; Lynn, T. Microservices migration patterns. Software: Practice and Experience, vol. 48(3), 2018, pp. 1-24.

Cloud Computing Patterns: load balancer. Disponível em: <http://cloudpatterns.org/mechanisms/load_balancer>. Acesso em 4 de outubro de 2018.

Evans E. Domain-Driven Design: tackling complexity in the heart of software. Boston, MA: Addison-Wesley, 2004.

Lewis J.; Fowler M. Microservices: a definition of this new architectural term. 2014. Disponível em <http://martinfowler.com/articles/microservices.html>. Acesso em 01 de setembro de 2018.

Fowler M. Strangler Application. 2004. Disponível em <https://www.martinfowler.com/bliki/StranglerApplication.html>. Acesso em 01 de setembro de 2018.

Humble J.; Farley D. Continuous delivery: reliable software releases through build, test, and deployment automation. Boston, MA: Addison-Wesley, 2010.

Jamshidi P.; Ahmad A.; Pahl C. Cloud migration research: a systematic review. IEEE Transactions of Cloud Computing, vol. 1(2), 2013, pp. 142-157.

Kratzke N.; Quint P-C. Understanding cloud-native applications after 10 years of cloud computing - a systematic mapping study. Journal of Systems and Software, vol. 126, 2017, pp. 1-16.

Newman, S. Building microservices: designing fine-grained systems. 1st ed. O'Reilly Media, 2015.

Santis S.; Flores L.; Nguyen D. V.; Rosa E. Envolve the monolith to microservices with Java and Node. 1st ed. Armonk, NY: IBM Redbooks, 2016.

Taibi, D.; Lenarduzzi, V.; Pahl, C. Processes, motivations, and issues for migrating to microservices architectures: an empirical investigation. IEEE Cloud Computing, vol. 4(5), 2017, pp. 22-32.

Vernon V. Implementing Domain-Driven Design. Addison-Wesley Professional. 1st ed. Pearson Education, 2013.