Microservice Architecture: Migrating A Point-of-Sale Application

Arquitetura de Microsserviços: Migração de Uma Aplicação de Ponto de Venda

Abstract. The microservice architecture has high popularity as it overcomes the shortcomings of monolithic architecture such as limited module scalability, arduous maintenance, technology dependency, etc. Therefore, it is crucial to migrating to the microservice architecture. Therefore, there is a need for a generic migration process, that is, that works in any scenario, this study presents the migration of a point of sale application to test a migration process declared as functional and generic.

Resumo. A arquitetura de microsserviços dispõe de popularidade elevada, uma vez que, supera as deficiências da arquitetura monolítica, como por exemplo: escalabilidade limitada a nível de módulo, manutenção árdua, dependência tecnológica, etc. Assim sendo, torna-se crucial migrar para a arquitetura de microsserviços. Logo, há a necessidade de um processo de migração genérico, isto é, que funcione em qualquer cenário, este estudo apresenta a migração de uma aplicação de ponto de venda com o intuito de pôr a prova um processo de migração declarado como funcional e genérico.

1. Introdução

A arquitetura monolítica fundamenta-se no encapsulamento integral das funcionalidades acerca de uma única aplicação que é posta em execução por meio de um único processo. Essa tem sido a arquitetura de maior predominância no âmbito de desenvolvimento de software. A ampla utilização da arquitetura monolítica não é à toa, uma vez que oferece uma solução que pode ser implementada de maneira simples. Além disso, conforme Chen et al. (2017) e Koschel et al. (2017) é facilmente dimensionada empregando instâncias da aplicação em companhia de um balanceador de cargas e a implantação também é descomplicada, visto que há somente um arquivo para ser implantado.

Os problemas da arquitetura monolítica surgem ao passo que a aplicação aumenta, eles causam diversos impactos como: alta acoplagem de códigos, compreensão dificultosa, manutenção árdua, lentidão ao decorrer do processo de compilação, acréscimo de esforço de comunicação entre envolvidos no projeto, etc. Além do mais, alguns empecilhos são herdados da própria natureza da arquitetura, por exemplo: desprovimento de escalabilidade a nível de módulo, dependência tecnológica e implantação contínua dificultosa.

À vista disso, objetivando prover soluções para as deficiências da arquitetura monolítica, manifestou-se a arquitetura de microsserviços. Conforme Fritzsch et al. (2015) e Taibi et al. (2017) essa é uma arquitetura nativa da nuvem, por esse motivo é notório a presença da propriedade de escalabilidade e disponibilidade, além de outras propriedades benéficas que a nuvem proporciona. Ainda assim, segundo Balalaie et al. (2015), apesar da constatação da escalabilidade em arquiteturas nativas da nuvem, para que de fato a aplicação torne-se dimensionável é necessário que a mesma esteja incorporada em uma arquitetura escalável, por exemplo, a arquitetura de microsserviços.

Thönes (2015), Yugopuspito et al. (2017) e Koschel et al. (2017), apontam que a arquitetura de microsserviços é decretada como um aglomerado de boas práticas que abrange várias características, a título de exemplo pode-se citar: DevOps, heterogeneidade tecnológica, escalabilidade, serviços autônomos, conteinerização, etc. Os principais componentes desta arquitetura são os microsserviços que, por sua vez, não possuem uma definição exata na literatura, porém de acordo com Baresi et al. (2017), Taibi et al. (2017), Yugopuspito et al.

(2017), Balalaie et al. (2015) e Koschel et al. (2017) são pequenos serviços autônomos que são capazes de desempenhar uma tarefa claramente definida de modo eficiente por via de processos individuais.

Conforme Chen et al. (2017) as aplicações construídas de acordo com a arquitetura de microsserviços possuem diversos aspectos vantajosos, tendo como principais a escalabilidade a nível de módulo, manutenção e implantação facilitada. Diante das vantagens apresentadas e outras não abordadas, diversas empresas como Netflix, SoundCloud e Amazon optaram por migrar da arquitetura monolítica para a de microsserviços.

Apesar disso, a migração não é classificada como um procedimento simples, pois de acordo com Thönes (2015) e Christoforou et al. (2017) envolve questões complexas que necessitam de estratégias para serem solucionadas. Dessa maneira, segundo Dragoni et al. (2017) a migração é considerada um tópico de alta sensibilidade na atualidade.

O presente artigo visa pôr a prova o processo de migração denominado P2, descrito por Taibi et al. (2017). A aplicação utilizada no processo de migração chama-se OpenPDV, cujo é, uma aplicação monolítica que tem como finalidade proporcionar aos usuários a efetuação de vendas de maneira mais eficiente, ágil e segura.

As informações expostas neste trabalho são de grande relevância, pois proporcionarão conhecimento a respeito da migração de aplicações monolíticas para microsserviços, além de servir como base para futuros estudos.

O restante deste estudo está organizado da seguinte maneira: seção 2, discute-se sobre os trabalhos relacionados; seção 3, aborda-se o estudo de caso; seção 4, aponta-se os resultados e a discussão; seção 5, explana-se a conclusão.

2. Trabalhos Relacionados

Ao longo do desenvolvimento do vigente estudo, foram constatados 3 trabalhos que discorrem a respeito da migração de aplicações monolíticas para a arquitetura de microsserviços. Assim sendo, a seção corrente tem como propósito descrever os trabalhos encontrados apontando como o tema é abordado por cada um deles.

O artigo elaborado por Yugopuspito, Panduwinata e Sutrisno (2017) retrata o processo de migração de uma aplicação de estacionamento que fundamenta-se na arquitetura monolítica. À vista disso, é sugerida uma abordagem para alcançar a arquitetura de microsserviços. Além disso, são revelados os impactos produzidos pela migração em relação ao negócio. Por fim, é concebida uma nova aplicação baseada na arquitetura de microsserviços e conclue-se que é necessário realizar experiências visando medir a eficácia dos microsserviços quanto a granularidade.

O estudo de Gouigoux e Tamzalit (2017) explana a migração de um monolito para a arquitetura de microsserviços enfatizando as lições técnicas assimiladas no decorrer do processo de migração. Dessa forma, 3 questões relacionadas a granularidade, implantação e orquestração são discutidas. Além disso, é exposto alguns benefícios provenientes da migração, como por exemplo: aumento de reutilização, substituição facilitada, aumento de desempenho, etc. No final, os autores destacam os benefícios alcançados e que a migração trata-se de uma questão técnica, metodológica e gerencial.

Já o trabalho desenvolvido por Dragoni et al. (2017) apresenta um estudo de caso baseado em um monolito crítico denominado FX Core que pertence ao Danske Bank, considerado o maior banco da Dinamarca. Os autores abordam o processo de migração para a arquitetura

de microsserviços concedendo destaque para a escalabilidade. Além disso, explanam os aspectos que devem ser atendidos para garantir a escalabilidade, por exemplo: automação, orquestração, balanceamento de carga, etc. Realizam também uma comparação entre as duas arquiteturas, tomando como exemplo o FX Core, onde são apontados fatores positivos e negativos.

Os trabalhos apresentados por Yugopuspito, Panduwinata e Sutrisno (2017), Gouigoux e Tamzalit (2017) embasam-se em caso real de migração, porém não descrevem claramente um processo de migração a ser seguido. Já Dragoni et al. (2017) também expõem um caso real de migração, entretanto propõem um processo de migração aplicado a partir do seu próprio cenário, não sendo classificado esse como processo generalizado.

Dessa forma, é importante observar a ausência de um processo de migração generalizado, funcional e comprovado. À vista disso, este estudo diferencia-se pelo fato de pôr a prova o processo de migração descrito por Taibi et al. (2017) que declara encaminhar uma aplicação monolítica para uma aplicação baseada em microsserviços por meio de cinco etapas: análise da estrutura do sistema, arquitetura do sistema, priorização do desenvolvimento dos serviços, codificação e testes. Logo, percebe-se que as informações resultantes poderão ser significativas para indivíduos interessados no tópico já que agregará conhecimento. Além disso, servirá também como base para posteriores estudos.

3. Estudo de Caso

O OpenPDV é uma aplicação de ponto de venda que contém funcionalidades requeridas pelo PAF-EFC (Programa Aplicativo Fiscal – Emissor de Cupom Fiscal), a título de exemplo, manipulação de nota fiscal eletrônica. Além disso, é uma aplicação baseada na arquitetura monolítica, fácil de instalar e de código aberto, ou seja, disponibiliza o código-fonte para estudar, modificar e distribuir. Levando-se em consideração esses aspectos, foi determinado que o estudo de caso seria baseado no OpenPDV.

Taibi et al. (2017) apotam 3 processos de migração generalizados que foram identificados a partir de entrevistas realizadas com alguns profissionais da área. Esses processos foram nomeados como P1, P2 e P3. Os dois primeiros processos preconizam a eleminição da aplicação monolítica e o nascimento da aplicação embasada na arquitetura de microsserviços. De maneira desigual, P3 recomenda a implantação dos microsserviços gradualmente, à medida que manifesta-se a necessidade de dispor novos recursos ou alterações em recursos existentes, substituindo-os por microsserviços.

À vista disso, P1, P2 e P3 conduzem para a arquitetura de microsserviços mediante cinco etapas que são intituladas de forma semelhante, embora possuam peculiaridades. As etapas podem ser analisadas na figura 1.

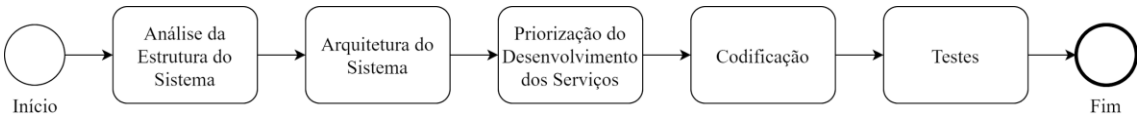

Figura 1. Etapas do processo de migração

A Análise da Estrutura do Sistema consiste na identificação das dependências da aplicação, ou seja, de que forma as entidades estão conectadas e como ocorrem as solicitações para aplicações terceiras. Além disso, P1 preocupa-se com o rastreamento de códigos que monopolizam o sistema realizando a maior parte das operações. Em função disso, essa etapa pode ser executada manualmente ou com o auxílio de ferramentas.

A Arquitetura do Sistema é a etapa em que ocorre a definição da estragéia para a dissociação do monolítico. Determina-se também uma coleção de princípios e diretrizes arquiteturais. Além de estabelecer quaisquer decisões que serão necessárias ser pactuadas entre os serviços apontados. Nessa etapa P1 indica uma arquitetura B que será colocada em vigor caso sucedam circunstâncias inesperadas e também realiza uma análise de alto nível a respeito dos riscos provenientes da adoção das diferentes arquiteturas.

A Priorização do Desenvolvimento dos Serviços é onde delibera-se a ordem de desenvolvimento dos serviços definidos, de acordo com o critério de prioridade indicado por cada processo de migração que está visível na tabela 1.

Tabela 1. Critério de prioridade dos processos de migração

Processo	Critério de Prioridade
P1	Julga-se a prioridade através da relevância que a funcionalidade detém para o cliente. Logo, quanto maior a relevância mais superior é a prioridade.
P2	A prioridade baseia-se em serviços que já existem. Sendo assim, leva-se em consideração o número de bug, dependência e esforço para atender ao cliente. Quanto menor a quantidade de dependência, maior é a prioridade. Para a contagem de bug, aplica-se o contrário, quanto maior o número de bug, maior é a prioridade. Caso a prioridade do cliente seja semelhante para dois ou mais serviços, olha-se sempre para o serviço que detém menor quantidade de dependência.
P3	Desenvolve-se apenas novos microsserviços que são empregados em volta do monolítico. A longo prazo os microsserviços irão gradualmente eliminando cada característica do monolítico.

A penúltima etapa é a Codificação que consiste na implementação dos microsserviços. Posteriormente, finalmente chega-se a etapa de Testes que é a última etapa dos processos de migração. Dessa forma, executa-se testes unitários e de integração. P2 e P3 também realizam testes de caixa preta.

A fim de encaminhar o OpenPDV para a arquitetura de microsserviços, utilizou-se o processo de migração intitulado P2 por aparentar ser mais adelgaçado e possuir menor nível de complexidade.

Em vista disso, realizou-se a primeira etapa com a assistência da ferramenta denominada SchemaSpy. Essa ferramenta é desenvolvida na linguagem de programação Java e requer a utilização do Java 5 ou versão superior. O SchemaSpy é capaz de conceber uma representação visual do esquema de banco de dados analisando-a de maneira automatizada.

A representação visual é confeccionada com a utilização de HTML (HyperText Markup Language) e JavaScript, tornando-se fácil o compartilhamento entre os interessados. Na figura 2 expõe-se o modelo lógico gerado a partir da ferramenta que visa demonstrar as dependências da entidade nomeada como venda, e.g., usuário, cliente, impressora, z, produto, troca e pagamento.

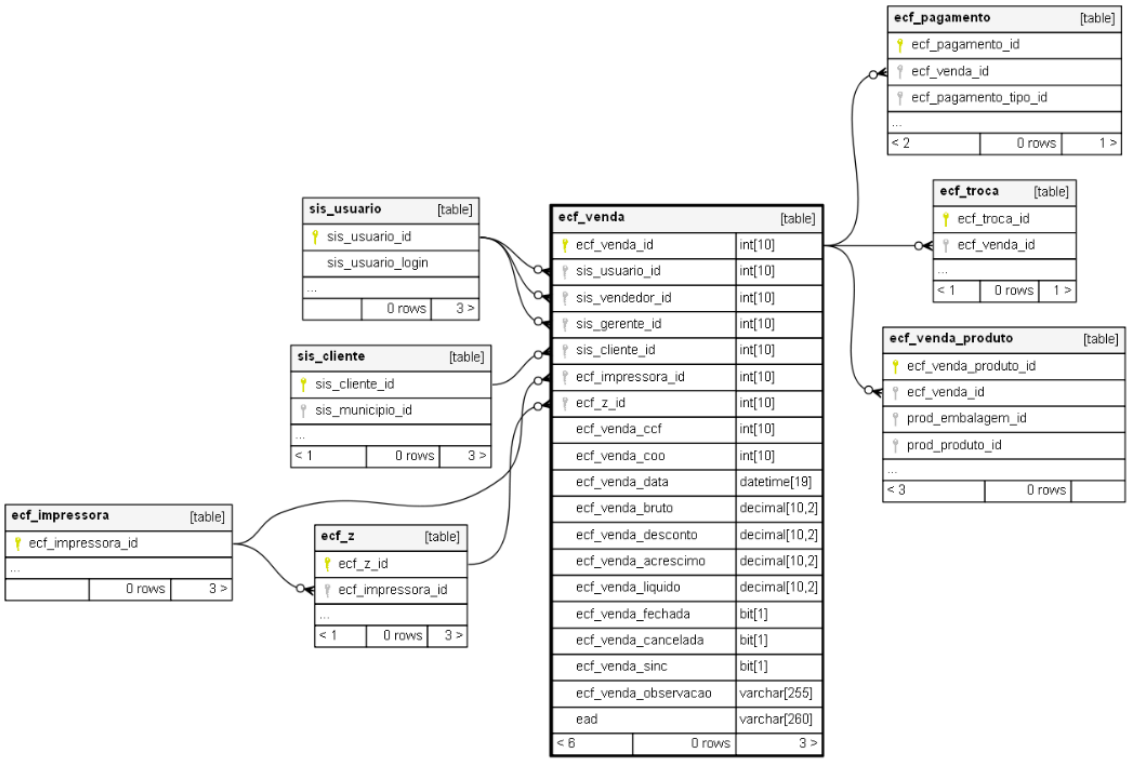

Figura 2. Modelo lógico: dependências da entidade venda

Dragoni et al. (2017) apresentam 3 princípios da arquitetura de microsserviços: contexto limitado, implica que os microsserviços devem conter somente funcionalidades que correspondem ao recurso no qual está sendo representado; dimensão reduzida, proporciona manutenibilidade e extensibilidade aos microsserviços; independência, provoca nos microsserviços a baixa acoplagem e alta coesão.

Dessarte, levando-se em consideração os princípios explanados, desempenhou-se a segunda etapa do processo de migração. A decomposição do monolítico foi executada cautelosamente e de modo manual. Além disso, como forma de acesso aos microsserviços, estabeleceu-se o estilo arquitetural de software denominado REST (Representational State Transfer). Por fim, 12 microsserviços foram identificados: venda, pagamento, cliente, usuário, produto, nota fiscal eletrônica, estado, município, empresa, impressora, troca e embalagem.

A priorização do desenvolvimento dos serviços ocorreu conforme o critério de priorização do processo de migração. Desse modo, observando-se o número de dependências de cada microsserviço, pode-se notar que a maior parte é equivalente, consequentemente a ordem de desenvolvimento deve ser apontada pelo cliente. Na tabela 2 é possível analisar a quantidade de dependências de cada microsserviço.

Levando em consideração que o OpenPDV é uma aplicação de ponto de venda, pode-se julgar o microsserviço de venda como sendo extremamente significante para o cliente. Assim sendo, coerentemente o microsserviço de venda detém prioridade maior.

Tabela 2. Número de dependências dos microsserviços

Microsserviço	Número de dependência(s)
Venda	5
Pagamento	1
Cliente	1
Usuário	0
Produto	1
Nota fiscal eletrônica	1
Estado	0
Município	1
Empresa	1
Impressora	0
Troca	1
Embalagem	0

A codificação sucedeu-se de acordo com a ordem de desenvolvimento estabelecida por meio da prioridade de cada microsserviço. É importante destacar que utilizou-se o Java e o Spring, uma vez que, os pesquisadores dispõem de afinidade com a linguagem de programação e o framework apontado. Nesta etapa, resolveu-se construir apenas o microsserviço de prioridade maior, já que a construção de todos os microsserviços exigiria uma infraestrutura de maior complexidade.

A última etapa do processo de migração deu-se por meio da ferramenta nominada Selenium. Essa ferramenta possibilita a execução de teste funcional de modo automatizado. Dessa forma, é possível assegurar a conformidade da aplicação com os requisitos funcionais.

4. Resultados e Discussão

O processo de migração P2 demonstrou-se efetivamente generalizado, como consequência pode ser aplicado em diversos cenários. Em comparação com P1 e P3, P2 pode ser considerado adelgaçado e com inferioridade em relação ao esforço necessário para encaminhar uma aplicação monolítica para a arquitetura de microsserviços, uma vez que, não exige numerosas tarefas ao longo da execução das etapas.

Ao decorrer da execução do processo de migração, nota-se que eliminar a aplicação monolítica para dá início a aplicação baseada na arquitetura de microsserviços pode ser um empecilho para que a migração ocorra de forma imperceptível para o cliente, já que, é necessário parar o monolito. Além disso, comprometerá a disponibilidade da aplicação.

Dessarte, para tornar a migração transparente e manter a disponibilidade da aplicação monolítica, recomenda-se realizar a migração de maneira gradativa sem suspender a execução do monolito. Essa é uma estratégia que pode ser adotada para solucionar o problema.

A complexidade do processo de migração encontra-se na execução das tarefas que compõem as etapas, por exemplo: dissociação do monolítico, é a tarefa que consiste na fragmentação do monolito em serviços, pode ser considerada relevante e difícil; definação da infraestrutura necessária para suprir a arquitetura de microsserviços é outra tarefa importante e que exige bastante conhecimento; realização de testes é mais uma atividade que deve ser realizada cuidadosamente para garantir o bom funcionamento dos microsserviços; etc.

No final do processo de migração obteve-se o microsserviço de venda, cujo é crucial para o OpenPDV, visto que trata-se de uma aplicação de ponto de venda. O microsserviço de venda é capaz de permitir a realização e o cancelamento de uma determinada venda. As operações mencionadas podem ser facilmente concretizadas por intermédio da invocação da interface REST.

O microsserviço apresentado no parágrafo anterior pode ser utilizado para fornecer o serviço de venda em aplicações mobile, web e desktop. Salientado que a regra de negócio encontra-se centralizada. Além disso, por conta dos princípios da arquitetura de microsserviços, há vantagens como escalabilidade, manutenibilidade e implantação facilitada.

Já em relação ao monolito a distribuição do serviço de venda ocorreria de forma descentralizada, ou seja, seria necessário reescrever a regra de negócio a cada aplicação desejada, por exemplo: web, desktop e mobile. À vista disso, torna-se possível notar uma possível complexidade na questão da manutenibilidade.

O código-fonte do microsserviço de venda pode ser consultado na plataforma de hospedagem de código-fonte denominada GitHub. O projeto encontra-se em um repositório Git público que está localizado no seguinte endereço: https://github.com/alexmouraable/openpdvms. Em companhia do projeto, segue o arquivo README.md que contém a documentação a respeito do serviço, isto é, apresentação dos endpoints, parâmetros, respostas, etc.

De forma semelhante a aplicação monolítica também encontra-se em um repositório Git situado no GitHub: https://github.com/pedrohlira/OpenPDV. O arquivo README.md contém um link para visualizar o funcionamento da aplicação por meio de um vídeo no youtube. Além disso, no projeto é possível encontrar as sentenças SQL que devem ser executadas para fazer a aplicação funcionar.

5. Conclusão

Neste estudo realizou-se a migração de uma aplicação monolítica nomeada OpenPDV para a arquitetura de microsserviços, por meio do processo de migração P2 descrito por Taibi et al. (2017) que declara encaminhar um monolito para uma aplicação baseada em microsserviços mediante cinco etapas: análise da estrutura do sistema, arquitetura do sistema, priorização do desenvolvimento dos serviços, codificação e testes.

Dessarte, foi possível descrever cada etapa e discorrer a respeito do processo de migração ao final da sua execução. Logo, pode-se inferir que as informações resultantes são significativas para indivíduos interessados no tema, uma vez que, agregará conhecimento. Além disso, servirá também como base para posteriores estudos.

References

Balalaie, A., Heydarnoori, A., & Jamshidi, P. (2015, September). Migrating to cloud-native architectures using microservices: an experience report. In European Conference on Service-Oriented and Cloud Computing (pp. 201-215). Springer, Cham.

Baresi, L., Garriga, M., & De Renzis, A. (2017, September). Microservices identification through interface analysis. In European Conference on Service-Oriented and Cloud Computing (pp. 19-33). Springer, Cham.

Chen, R., Li, S., & Li, Z. (2017, December). From monolith to microservices: a dataflow-driven approach. In 2017 24th Asia-Pacific Software Engineering Conference (APSEC) (pp. 466-475). IEEE.

Christoforou, A., Garriga, M., Andreou, A. S., & Baresi, L. (2017, November). Supporting the decision of migrating to microservices through multi-layer fuzzy cognitive maps. In International Conference on Service-Oriented Computing (pp. 471-480). Springer, Cham.

Dragoni, N., Dustdar, S., Larsen, S. T., & Mazzara, M. (2017). Microservices: Migration of a mission critical system. arXiv preprint arXiv:1704.04173.

Fritzsch, J., Bogner, J., Zimmermann, A., & Wagner, S. (2018, March). From monolith to microservices: a classification of refactoring approaches. In International Workshop on Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment (pp. 128-141). Springer, Cham.

Gouigoux, J. P., & Tamzalit, D. (2017, April). From monolith to microservices: Lessons learned on an industrial migration to a web oriented architecture. In 2017 IEEE International Conference on Software Architecture Workshops (ICSAW) (pp. 62-65). IEEE.

Koschel, A., Astrova, I., & Dötterl, J. (2017, July). Making the move to microservice architecture. In 2017 International Conference on Information Society (i-Society) (pp. 74-79). IEEE.

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

Taibi, D., Lenarduzzi, V., Pahl, C., & Janes, A. (2017, May). Microservices in agile software development: a workshop-based study into issues, advantages, and disadvantages. In Proceedings of the XP2017 Scientific Workshops (p. 23). ACM.

Thönes, J. (2015). Microservices. IEEE software, 32(1), 116-116.

Yugopuspito, P., Panduwinata, F., & Sutrisno, S. (2017, October). Microservices architecture: case on the migration of reservation-based parking system. In 2017 IEEE 17th International Conference on Communication Technology (ICCT) (pp. 1827-1831). IEEE.