Migração de Sistemas Monolíticos para Microserviço: Uma Abordagem Híbrida sem Downtime

Augusto Flávio Freire[1], Américo Sampaio[1], Otávio Medeiros[1], Nabor Mendonça[1]

[1]Programa de Pós-Graduação em Informática Aplicada (PPGIA)
Universidade de Fortaleza
Av. Washington Soares, 1321, Edson Queiroz, CEP 60811-905, Fortaleza, CE, Brasil

augusto.arraes@ifce.edu.br, americo.sampaio@unifor.br,
otaviomd@hotmail.com, nabor@unifor.br

Abstract. *Several organizations need to address the challenge to migrate current traditional monolithic applications in production to microservices, preferably, without having to schedule maintenances to take the application offline. This paper presents an approach for hybrid migrating to microservices with no downtime and practically no changes in the monolithic code. A real application is used as a proof of concept to demonstrate that the proposed approach enables to go "forward" or "backward" among different versions of the application without needing to take the application offline, considering code or data changes. An evaluation performed in the cloud demonstrates that this work does not introduce significant overhead.*

Resumo. *Várias organizações precisam solucionar o desafio de migrar as aplicações monolíticas tradicionais em produção para microserviço, de preferência, sem ter que causar manutenções programadas para retirar a aplicação do ar. Este artigo apresenta uma abordagem de migração híbrida para microserviços sem downtime e praticamente sem alteração de código do monolítico. Uma aplicação real é utilizada como prova de conceito para demonstrar que a abordagem proposta permite "ir" e "voltar" entre diferentes versões da aplicação sem a necessidade de retirar o sistema do ar, considerando modificações de código ou dados. Uma avaliação realizada na nuvem demonstra que este trabalho não introduz overhead significativo.*

1. Introdução

Microserviços vem se tornando uma das principais estratégias para construir aplicações distribuídas [Richardson 2017], [Newman 2015], [Balalaie et al. 2015a]. Com o paradigma de pequenas aplicações independentes, autônomas e escaláveis, os microserviços podem ser implantados e distribuídos de forma integrada e transparente ao usuário como se fosse uma única aplicação. Os maiores provedores de serviços do mundo, como Amazon, Netflix, Google e outros já utilizam o conceito de microserviços em suas aplicações.

Os microserviços vem inovando alguns conceitos da engenharia de software de como os sistemas estão sendo construídos, diferente dos tradicionais monolíticos. Diante dessa nova tecnologia de construir sistemas (independentes, escaláveis e integrados), há um grande desafio para o mercado e a comunidade acadêmica em migrar as aplicações monolíticas para microserviços. Comumente, as aplicações monolíticas já se encontram há um bom tempo em ambiente de produção sendo utilizadas por clientes,

já receberam várias atualizações e armazenam dados em bases de dados. Sendo assim, não é uma tarefa trivial migar estes monolíticos para microserviços, pois pode-se necessitar significativas alterações em código e nos dados armazenados necessitando de interrupções programadas para manutenção nos sistemas.

Outro fator complementar a se considerar numa possível migração do monolítico para microserviço é a escalabilidade da aplicação. Se um determinado módulo do monolítico é sobrecarregado, a aplicação inteira deverá ser replicada para atender a demanda das requisições pelo fato do monolítico possuir uma única unidade de implantação. Caso este determinado módulo fosse migrado para microserviço, apenas ele poderia ser replicado para escalar pois iria conter sua própria unidade independente de implantação. Esse é um dos fatores que motiva as organizações em migrar suas aplicações web tradicionais para microserviços podendo usufruir da escalabilidade independente e economia de recursos computacionais.

Migrar as aplicações monolíticas tradicionais para microserviço é um desafio de pesquisa em aberto. Decidir o que migrar, quando migrar e como migrar do monolítico para microserviço é uma tarefa complexa principalmente para monolíticos já em produção.

Este artigo apresenta uma abordagem híbrida de migração para microserviços sem *downtime* e praticamente sem alteração de código. Consideramos nossa abordagem híbrida pelo fato de que adotamos a estratégia de manter o monolítico em produção funcionando em conjunto com os microserviços que vão sendo desenvolvidos paulatinamente para substituir partes da funcionalidade dos módulos do monolítico. Para que possamos realizar as migrações sem *downtime* e praticamente sem alterações no código do monolítico adotamos conceitos de "*Reflection*" e Programação Orientada à Aspectos (AOP) que permite desviar chamadas à determinados módulos do monolítico (*Advice Around* da AOP) transformando-as em requisições REST (*Representational State Transfer)* ao código dos novos microserviços desenvolvidos.

Uma aplicação monolítica real é utilizada como prova de conceito para demonstrar que a abordagem proposta permite "ir" e "voltar" entre diferentes versões da aplicação sem a necessidade de retirar o sistema do ar, considerando modificações de código ou dados. Além disto, uma avaliação realizada na nuvem demonstra que a abordagem não introduz *overhead* significativo do ponto de vista de desempenho e nem de custos, e ainda permite suportar cargas maiores no monolítico devido à retirada de parte da funcionalidade do monolítico para ser processada em microserviço.

O presente trabalho está organizado da seguinte forma: A seção 2 descreve sobre o atual estado da arte de migração de aplicações monolíticas para microserviço. A seção 3 explica a abordagem de migração proposta neste trabalho, enquanto a seção 4 demonstra a avaliação de desempenho e de custo. A seção 5 apresenta a conclusão e os trabalhos futuros.

2. Migração de Aplicações Monolíticas para Microserviço

Eisele (2016) define três abordagens de migração para microserviço. A primeira, e menos arriscada, é migrar os serviços do monolítico que se adaptam melhor ao microserviço. É uma migração gradativa a cada serviço e os microserviços continuam associados aos respectivos serviços do monolítico. Em relação à proposta deste artigo, o microserviço se mantém de certa forma associado ao serviço do monolítico, pois no ato

da migração desacoplada essa propriedade é importante na chamada do microserviço correspondente. A outra abordagem de Eisele (2016) é semelhante a essa, mas o monolítico e o microserviço operam independentes. O monolitico não modifica os dados do microserviço, pois ocorre a migração da camada de dados e o microserviço mantém sua própria persistência, isolando a parte correspondente do monolítico. A terceira abordagem, a mais arriscada, sugere realizar a migração total para microserviços (*Big Bang Rewrite*). No entanto, é a abordagem menos recomendada para migração conforme afirma Eisele (2016), pois a disponibilidade dos serviços ainda poderá ser oferecida pelo monolítico. Já a proposta deste trabalho apresenta uma migração sem interrupções do monolítico. Sem precisar praticamente alterar código e sem realizar desligamentos programados ao sistema monolítico.

Outras abordagens de migração foram analisadas. Balalaie et al (2015b), Newman (2015), Richardson e Floyd (2016) tomam como estratégia de migração através da refatoração do monolítico identificando entidades de domínio e assim migrá-las para microserviços. Também Richarson (2014), Fowler e Lewis (2014) sugerem migrar para microserviços através de parâmetros convencionais da engenharia de software como tamanho do módulo, subdomínio da aplicação, ou relacionamento de funcionalidades e dados.

Uma abordagem semelhante ao trabalho deste artigo é do Richardson (2016b). O diferencial é que a abordagem deste artigo consegue voltar à versão anterior monolítica sem recompilar código, sem *downtime*, em tempo de execução. Outro trabalho semelhante que também adota técnica desacoplada de migração é do Bolar (2018). Esse utiliza *annotation* do Java, mas de toda forma altera o código do monolítico.

Medeiros (2018) realizou a migração de um serviço do monolítico para microserviço utilizando programação orientada a Aspectos, para interceptar a chamada do serviço no monolítico e encaminhar via REST para um microserviço correspondente atender a requisição solicitada. Essa abordagem é semelhante ao processo da "Ida" desta proposta de migração, ou seja, a realização da migração de fato do monolítico para o microserviço. Esta proposta de trabalho contribuiu com o caminho da "Volta", desfazendo a migração, caso isso seja decidido. Além de realizar a "Ida" e a "Volta" na camada de serviço (migração com Aspecto via REST), este trabalho também contribuiu com a "Ida" e com a "Volta" na camada de dados (microserviço com seu próprio banco de dados na "ida" e *recovery* dos dados do micro-banco ao banco monolítico na "volta"). Todo o processo ocorre praticamente sem alteração de código e sem *downtime*.

Diante da análise dessas abordagens de migração de aplicações monolíticas para microserviços, elaborou-se os principais requisitos (RM – Requisitos de Migração) necessários para avaliar uma abordagem de migração:

- [RM01] Capacidade de Refatoração: O quanto se pode refatorar um sistema monolítico em partes para ser migrado para microserviço. A refatoração poderia ser por módulo, por subdomínio do problema, por funcionalidades ou por serviço. Ou ainda, se é migrar parcialmente o monolítico [Balalaie et al. 2016], [Yanaga 2017], [Fowler e Lewis 2014] e [Newman 2015] ou refatorar por completo *(Big Bang rewrite)*.
- [RM02] Migração de Código: Depois de definir qual estratégia de refatoração será tomada, deve ser realizada a migração da parte do código do monolítico

para o código do microserviço. O microserviço deve atender e responder da mesma forma que o módulo correspondente no monolítico operava.

- [RM03] Alteração de Código: O quanto de código precisa ser alterado no sistema monolítico para realizar uma migração para microserviço, visto que segundo Eisele (2016) não é recomendado uma migração completa do monolítico para microserviço.
- [RM04] Associação entre Serviços: Que tipo de associação ou dependência existe durante e após a migração entre o que foi refatorado do monolítico com o microserviço correspondente [Eisele 2016].
- [RM05] Coexistência de ambos os paradigmas: Como seria a distribuição do monolítico e os microserviços, e também as respectivas bases de dados. Durante o processo de migração, ambos os serviços são balanceados para atender as requisições e qual dos serviços está responsável por atendê-las diante de quais estratégias a migração foi projetada.
- [RM06] Interrupção do Serviço: Na abordagem de migração, se deve interromper ou não o serviço do monolítico para realizar a migração para microserviço.
- [RM07] Migração de dados: Quando se trata sobre migração para microserviço, a primeira idéia é substituição de parte do código do monolitíco por código do microserviço. A migração de dados também deve ser um requisito, pois o microserviço encapsula ou está responsável por usa própria base de dados.
- [RM08] Caminho da "Volta": O fato de migrar para microserviço corresponde ao caminho da “Ida”. Mas se a migração apresentar problemas, ou se não corresponder como esperava, ou simplesmente se for uma migração temporária para atender uma certa demanda de requisições, de qualquer forma se deseja voltar para o monolítico também deve ser um requisito de migração para microserviço. Não há garantia de que todo o monolítico será migrado e também se o que foi migrado será definitivo. De toda forma, deve ser previsível a capacidade de voltar para o monolítico, pois este requisito pode ser bastante útil numa tomada de decisão necessária para desfazer a migração. E, se “Voltou”, após ajustes ou correções, pode realizar uma nova migração.

Esse último requisito não foi tratado nas abordagens de migração analisadas. Neste trabalho, além da migração sem *downtime* (desacoplada), foi proposto uma solução de “Voltar” para o monolítico, de forma transparente na operação da aplicação e sem interromper o serviço. Ao mesmo tempo, esse último também é um requisito proposto neste trabalho.

3. Abordagem Proposta

Dado um sistema monolítico que está sobrecarregado pela alta demanda de requisições por um determinado serviço, o qual necessita ser migrado para microserviço, há uma preocupação de como executar essa migração sem derrubar ou interromper a atividade do Monolítico. Essa questão é a motivação deste trabalho, cuja proposta é realizar uma migração parcial do Monolítico para o microserviço com zero *downtime*, com a capacidade de “ir” e “voltar”, ou seja, de poder ser migrado e também poder ser

retornado ao Monolítico (desfazer a migração), tudo em tempo de execução. Essa migração é de forma desacoplada, sem alteração de código do monolítico.

O trecho de código abaixo, Figura 1, mostra o Aspecto em Java que faz intercepção da chamada de serviço a ser migrado. Esse compara o serviço com os serviços do monolítico identificados pelo *Reflection* do Java, se o determinado serviço foi o escolhido para fazer a migração (seja um REST GET ou POST, por exemplo), a requisição HTTP (*HyperText Transfer Protocol)* é encaminhada ao microserviço. Essa técnica proporcionou a migração desacoplada e sem alterar o código original do monolítico.

```
@Around("bean(*Service) || bean(*Controller)")
public Object getIdMicroservice(ProceedingJoinPoint proceedingJoinPoint) throws Throwable {
    Object microservice = null;
    if( this.getIdMicroservice(proceedingJoinPoint.getSignature()) ) {
        System.out.println(".. @MicroserviceGatewayAOP call Microservice via REST: "+ "http://"+this.URL_BASE+":"+this.URL_PORT+"/" + SERVICE_NAME );
        // REST Method HTTP = GET
        if( this.discoveryMethodHTTP(proceedingJoinPoint.getSignature().toString()).equals("GET") )
            microservice = (Object) restTemplate.getForObject("http://"+this.URL_BASE+":"+this.URL_PORT+"/" + SERVICE_NAME, Object.class);
        // REST Method HTTP = POST
        if( this.discoveryMethodHTTP(proceedingJoinPoint.getSignature().toString()).equals("POST") ) {
            HttpHeaders httpRequestHeaders = new HttpHeaders();
            httpRequestHeaders.set("Content-Type", "application/json");
            HttpEntity<Object> request = new HttpEntity<Object>(proceedingJoinPoint.getArgs()[0], httpRequestHeaders);
            System.out.println(".. @OwnerAOP   restTemplate.postForObject");
            microservice = (Object) restTemplate.postForObject("http://"+this.URL_BASE+":"+this.URL_PORT+"/" + SERVICE_NAME, request, Object.class);
        }
    }else { // continua no monolitico a chamada do serviço
        System.out.println(".. @MicroserviceGatewayAOP prodeed, service call continues on Monolithic ");
        microservice = (Object) proceedingJoinPoint.proceed();
    }
    return microservice;
}

private boolean getIdMicroservice(Signature methodSignature) {
    String sig=methodSignature.toString();
    System.out.println(".. @MicroserviceGatewayAOP methodSignature: "+sig);
    boolean isMicroservice = false;
    for (String keyvalueMonolithicServices:DynamicProperties.monolithicServiceMethods) {
        if ( sig.substring(sig.lastIndexOf('.')+1,sig.lastIndexOf('(')).equals(
                keyvalueMonolithicServices.substring(keyvalueMonolithicServices.lastIndexOf('.')+1,keyvalueMonolithicServices.length()) ) ) {
            isMicroservice = this.loadKeyValueMonolithicService(keyvalueMonolithicServices);
            System.out.println(".. @MicroserviceGatewayAOP Target getIdMicroservice() : "+keyvalueMonolithicServices+" = "+isMicroservice);
            // Faz VOLTA
            mhc.goBackToMonolithic( keyvalueMonolithicServices, isMicroservice, "http://"+this.URL_BASE+":"+this.URL_PORT);
            return isMicroservice;
        }
    }
    return isMicroservice;
}
```

Figura 1. Migração para Microserviço desacoplada pelo Aspecto.

Para demonstrar a migração da abordagem proposta, um sistema Monolítico terceiro, desenvolvido em Java, utilizando o *Framework* Spring Boot [Macero 2017], foi escolhido: o PetClinic [Syer 2015]. O critério de escolha para o Monolítico era que estivesse escrito e organizado seguindo as boas práticas de desenvolvimento de sistemas em Java e de orientação a objetos. Outro fator também importante para a escolha, foi a disponibilidade do mesmo sistema em versão de Microserviço [Rey 2016], contemplando o *RM2*. Esse exemplo de aplicação, tanto em versão Monolítico quanto em versão Microserviço, foi adotado como ambiente de prova de conceito na migração híbrida (parcial e desacoplada) do Monolítico para um Microserviço, que correspondente ao um mesmo serviço do Monolítico. A migração do monolítico para o microserviço ocorre com zero *downtime e* sem alteração de código. E, caso a migração não seja mais necessária ou que já atendeu ao propósito, o serviço retorna ao Monolítico em tempo de execução (*RM08*), também sem causar *downtime* do mesmo.

Para esta abordagem, três fatores de migração são considerados quando se trata de migração híbrida de Monolítico para Microserviço, levando em questão "Ir" e "Voltar":

- Migração parcial: Algumas partes do monolítico são migradas para microserviço e não o monolítico completo (requisito *RM01*). Um ou mais serviços do monolítico são candidatos à migração, ou seja, há serviços que continuam sendo demandados para o monolítico e outros são migrados para microserviço (*RM05*).

Esses podem ser desativados e retornados ao Monolítico (*RM08,* caminho da "Volta") ou ainda podem ser migrados novamente em uma outra demanda.

- Migração Desacoplada: Não há necessidade de modificar o código do Monolítico (*RM03*) para realizar migração, seja na "Ida" ou na "Volta".
- Migração Zero *Downtime*: Durante todo o processo de migração, na "Ida" ou na "Volta", o monolítico não é desligado (*RM06*).
- Migração Micro-Banco: No conceito de Microserviço, cada um mantém a sua própria base de dados, diferente do Monolítico, no qual a base única é compartilhada para todos os serviços da aplicação. Então, atendendo essa premissa, ao migrar para microserviço, a entidade e os dados são extraídos (decompostos) da base monolítica e migrados para micro-base do microserviço correspondente. Assim as mudanças de estado de parte da aplicação serão mantidas pelo microserviço após a migração. Caso seja desfeita a migração, os dados serão atualizados na base monolítica (*RM07*).

3.1. Estratégia de Migração Híbrida de Sistema Monolítico para Microserviço

Numa migração híbrida de sistema monolítico para microserviço, para a abordagem proposta, deve-se levar em consideração os requisitos de migração e os pontos mencionados no item anterior: parcial, desacoplada, zero *downtime* e micro-banco. Tudo isso para tornar a migração híbrida de forma que possa "Ir" e "Voltar".

Na Figura 2, tem-se a modelagem da estratégia proposta. Nota-se que não há uma substituição definitiva de Monolítico para Microserviço (*RM01*). O contexto é de coexistência de ambos os modelos ainda como se fosse uma única aplicação (*RM05*). A implantação do Microserviço é de troca de requisição entre o Monolítico (*RM02*), de forma desacoplada, sem alterá-lo em código (*RM03*), e sem desligá-lo, com zero *downtime* (*RM06*), pois os demais serviços continuarão sendo atendidos pelo Monolítico (*RM05*).

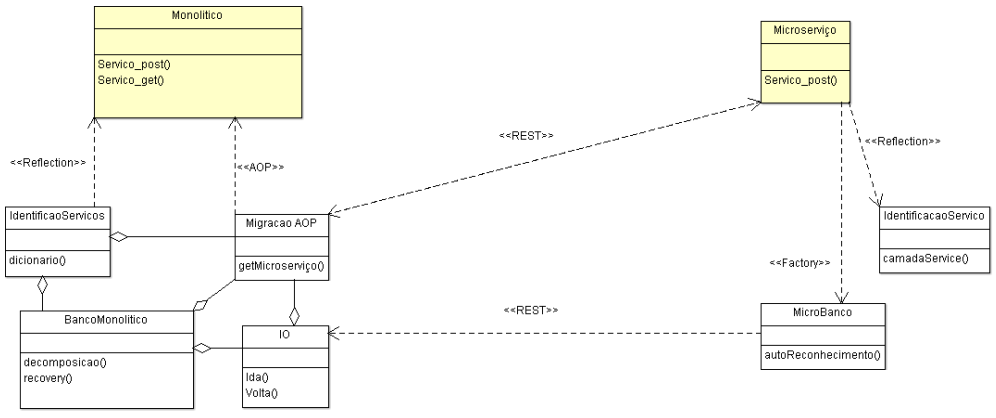

Figura 2. Estratégia proposta de Migração Híbrida.

De acordo com a estratégia proposta, a identificação de serviços é fundamental na camada de aplicação (*RM01*) e na camada de persistência. Na camada de aplicação, tem-se a visão do que será escolhido para ser migrado para Microserviço e do que permanecerá no Monolítico. O levantamento de informações mapeadas com as

dependências entre a camada de aplicação e de persistência estão disponibilizadas em todo o contexto para a migração do lado Monolítico.

Todos esses dados complementam a principal interface do padrão, a migração via AOP [Kiczales 1997]. É a interface de migração de fato entre o Monolítico e o Microserviço. De forma desacoplada, requisições HTTP ao Monolítico são interceptadas pelo Aspecto e encaminhadas via REST ao Microserviço (*RM02*), sem alteração de código do monolítico (*RM03*), o qual torna possível a não interrupção do monolítico para migrar a requisição (*RM06*). O serviço monolítico que será interceptado e como será executado (*RM04*) são garantidos pela identificação de serviços [Levcovitz et al. 2015] [Richardson 2016b]. Uma vez desacoplado, da mesma forma que foi migrado, pode ser desativado sem prejuízos à aplicação (*RM08*).

Durante a etapa de migração, como previsto no padrão, a criação do micro-banco (camada de persistência do Microserviço, *RM07*) é gerenciado do lado Monolítico. Essa criação do micro-banco é a decomposição da entidade (ou caso for mais de uma) e respectivos dados da base monolítica referente ao mapeamento do serviço-entidade. O micro-banco também é fatorado de forma desacoplada. Então, quaisquer alterações de dados, após a migração, na decisão de desfazer a migração (*RM08*), a "Volta", é garantida as diferenças de estado na base monolítica, de forma transparente e sem *downtime*. Uma mensagem via REST entre os paradigmas, Monolítico e Microserviço, através também do Aspecto, ao escutar que a migração foi desligada, é acionada para fazer o *recovery* dos dados do micro-banco para a base do monolítico.

Os elementos do padrão proposto foram elencados para garantir a "ida" (*RM01 a RM04*), de forma híbrida (*RM05*), e a "volta" (*RM08*) entre diferentes versões da aplicação sem a necessidade de retirar o sistema do ar, *RM06*.

3.4. Caso PetClinic: Cenário de Migração Híbrida do Sistema Monolítico para Microserviço

Uma API (*Application Programming Interface*) foi desenvolvida e implantada para sistema monolítico e outra distribuição voltada para microserviço também foi escrita. A API foi implantada no Monolítico PetClinic e também em um dos Microserviços do PetClinic. A API foi desenvolvida em Java baseada em Spring Boot.Tudo parte de uma escolha binária de uma lista de módulos do monolítico que possam ser migrados para microserviço. Essa lista é auto-identificada pela API. Uma vez escolhido o serviço, como Liga/Desliga, o serviço estará apto para ser migrado para microserviço. A migração ocorre sem interromper qualquer atividade do Monolítico. Utilizando programação orientada a Aspecto (AOP), a API intercepta a requisição da chamada da camada de serviço e de imediato encaminha ao referente microserviço via REST.

Enquanto estiver "*true*", a migração estará ocorrendo, em tempo de execução, a cada chamada nesse determinado serviço. Os demais serviços continuam sendo atendidos normalmente pelo Monolítico, a não ser que, algum outro também possa ser escolhido para ser migrado, bastando que: (1) o respectivo microserviço esteja desenvolvido e disponibilizá-lo, e (2) seja ativado, *true*", para ser também migrado. Essa decisão pode ser a qualquer momento, pois não é necessário programar interrupções no Monolítico para fazer a migração, sem *downtime*. A arquitetura da aplicação, durante essa migração parcial e desacoplada, torna-se híbrida, pois os

serviços da aplicação serão atendidos pelo monolítico e hora por um ou mais microserviços.

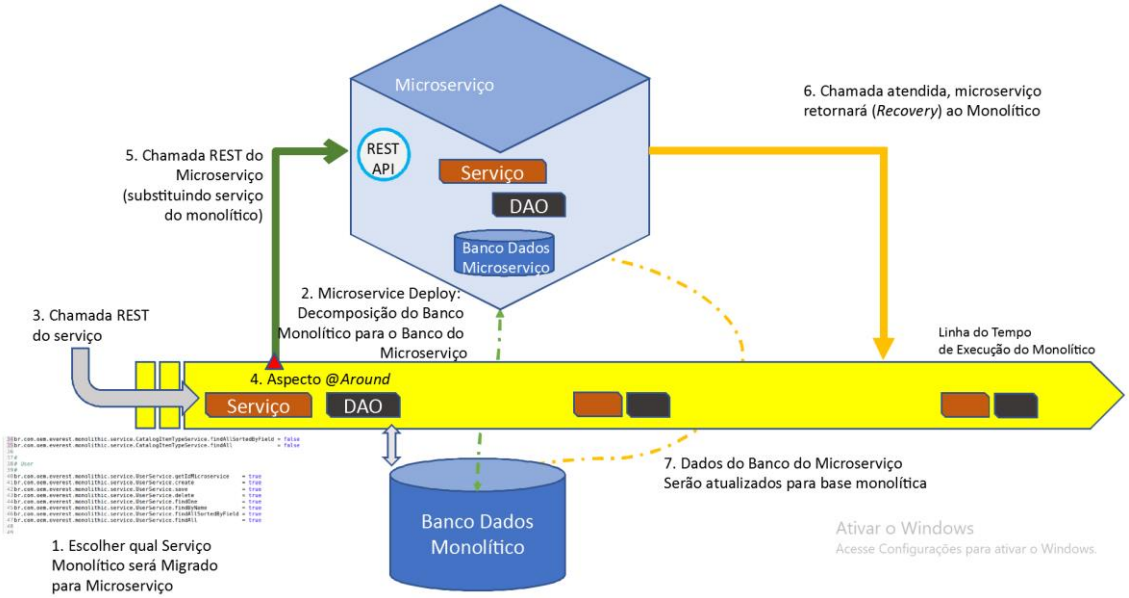

Figure 3. Ciclo de Migração Híbrida do Monolítico para Microserviço.

Todas essas operações ocorrem em tempo de execução, sem *downtime* e sem alteração de código. Conforme a Figura 3, observa-se que ao ser migrado o serviço monolítico para microserviço, também ocorre a migração da base de dados. Até o momento, verificou-se a migração na camada de aplicação. A API também realiza a migração na camada de dados, conforme previsto na estratégia.

No ato da migração e na primeira vez que o microserviço responder pelas chamadas do serviço via monolítico, a API decompõe o que o serviço utiliza de entidade na base de dados monolítica e cria em tempo de execução o micro-banco do microserviço. Neste momento, a mesma entidade da base monolítica não sofrerá mudanças de estado, pois isso será da responsabilidade do microserviço, já que o mesmo está atendendo as chamadas da migração via monolítico. Todas essas etapas de migração na camada de serviço e na camada de dados correspondem ao caminho da "Ida" da migração híbrida proposta neste trabalho.

Para o caminho da "Ida" no PetClinic, foi escolhido o serviço de cadastro de cliente e para microserviço também o mesmo serviço correspondente. Isso é uma simetria importante para migração entre o monolítico e o microserviço. Todas as requisições de cadastro solicitadas ao monolítico foram interceptadas via Aspecto e encaminhadas ao microserviço. Os dados já foram cadastrados no micro-banco desse microserviço, depois da decomposição da base monolítica da respectiva entidade do serviço. Um vídeo demonstrativo da API em funcionamento no PetClinic está disponibilizado [Freire 2019]. O vídeo demonstra tanto o caminho da "Ida" quanto o da "Volta".

Quanto à "volta", a decisão é se será "desligada" a migração. Essa decisão não afeta nenhuma interrupção da aplicação. Então ao ser desabilitada a migração, escolhendo "*false*", na próxima requisição ao serviço migrado ao monolítico, a API via Aspecto ainda a interceptará, mas não encaminha essa requisição ao microserviço.

Desse ponto em diante, o ambiente da aplicação, em tempo de execução, deixará de ser híbrido e ficará monolítico como antes, caso não haja mais nenhum módulo migrado a ser desligado.

Assim como na camada de serviço teve seu retorno para o monolítico, na "volta" na camada de dados também acompanha no mesmo retorno. Antes da requisição ser repassada de volta ao próprio monolítico, os dados do micro-banco serão refletidos para o banco do monolítico, caso tiverem sidos alterados. Quaisquer dados inseridos, alterados ou excluídos durante a migração serão refletidos no banco monolítico na "volta".

Na "Volta" no PetClinic, os dados que foram cadastrados na migração pelo microserviço foram refletidos na base monolítica. Isso acontece ao desligar a migração. Os dados da próxima requisição ao serviço do monolítico nesse momento já foram persistidos na base monolítica, logo após a atualização dos dados do micro-banco à base de dados monolítica. No mesmo vídeo, foi demonstrado também a "Volta" no PetClinic.

Concluída a volta, o monolítico mantém-se o mesmo devido à migração desacoplada, sem *downtime* e seu respectivo banco atualizado, de forma transparente ao usuário. A partir desse ponto, o monolítico poderá iniciar novo ciclo de migração híbrido.

4. Avaliação

Uma avaliação realizada na nuvem, Amazon AWS, demonstra que este trabalho não introduz *overhead* de desempenho e nem de custos significativos e ainda permite suportar cargas maiores no monolítico, devido à retirada de parte da sua funcionalidade para ser processada em microserviços.

Para a realização dos testes, foram escolhidos três serviços REST do Monolitíco PetClinic [Syer 2015], no qual 70% das requisições se concentram no serviço com método POST e os 30% restantes com as requisições dos outros serviços com método GET. O serviço com maior proporção de chamadas foi o candidato a microserviço. No entanto, os testes foram realizados em duas perspectivas de topologias: Homogênea (todas as requisições são realizadas ao Monolítico - forma tradicional) e Híbrida (todas as requisições chegam ao Monolítico, mas as requisições de um serviço POST representando 70% das requisições do teste são migradas para o microserviço, desacoplada via Aspecto; e, as demais requisições, GET dos outros dois serviços do teste, são atendidas pelo Monolítico).

Em ambas as topologias, um teste de carga foi aplicado durante 6 minutos, com 100 usuários-teste por 10 segundos submetendo, cada um, 10 requisições HTTP (via REST) sobre o Monolítico. Conforme proporção anterior, das 10 requisições, 7 são de cadastro de cliente (método POST) e 3 de visualizações (GET) de veterinários. Então, por cada minuto, uma carga de 60 mil requisições foi aplicada sobre a aplicação no monolítico (modo homogêneo) e depois, no modo híbrido, o monolítico recebendo todas as mesmas requisições, mas encaminhando ao microserviço as requisições POST (70%). Uma avaliação de desempenho e de custo foram realizadas nesse cenário.

4.1. Desempenho

Aplicado os testes, gerou-se métricas de uso de CPU, memória e tempo de resposta.

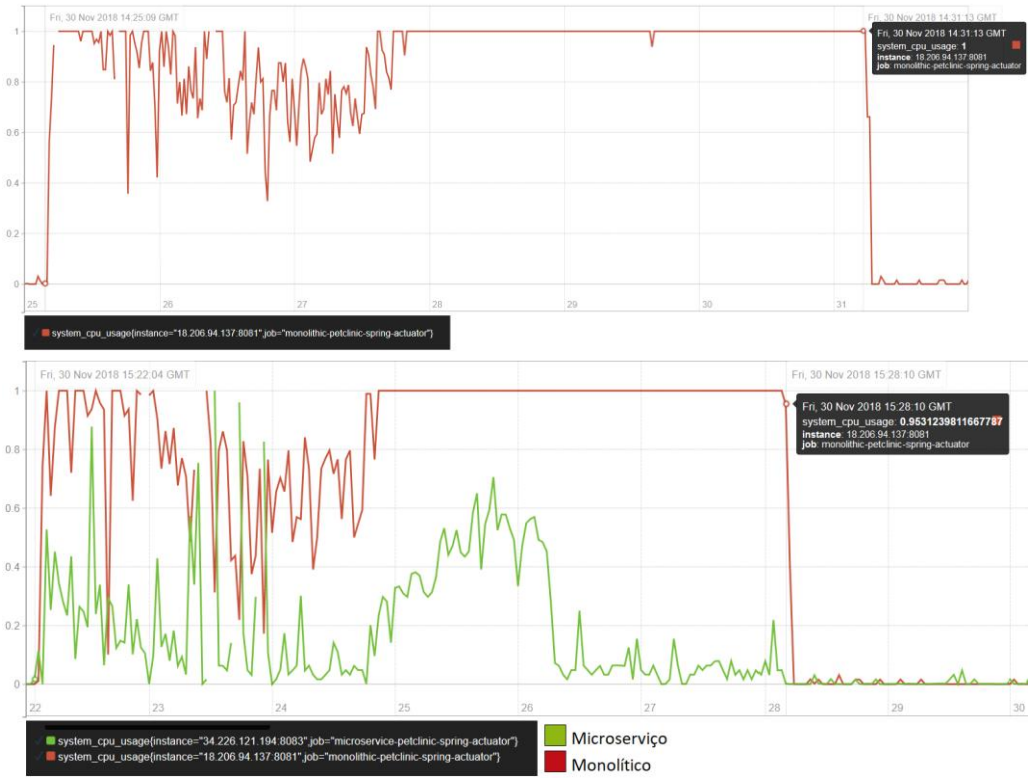

Figure 4. Consumo de CPU: Homogêneo e Híbrido.

No primeiro ciclo de testes, com 100 requisições, de acordo com a Figura 4, nota-se que na abordagem híbrida, com microserviço, que o esforço do monolítico chega a ser por volta de 20%, enquanto microserviço assume o maior esforço quando é responsável por atender o serviço requisitado. Enquanto, no primeiro gráfico, comparando na mesma ordem de tempo, o monolítico consumiu 60% (pouco antes do 3º. minuto). Outra observação é que num dado momento do teste de carga, nas duas perspectivas (tanto homogênea quanto híbrido), o consumo de CPU do monolítico chega ao máximo. No entanto, ainda assim, no híbrido, nota-se que o microserviço continua atendendo as requisições migradas do monolítico.

Figure 5. Consumo de memória: Homogêneo e Híbrido.

Tanto na perspectiva homogênea quanto na híbrida, a atividade de memória (Figura 5) manteve-se praticamente a mesma. Em ambas as situações, vale ressaltar que todas as requisições chegam ao monolítico. A diferença que na híbrida é possível visualizar a atividade de memória próximo a do homogêneo (quando apenas está o monolítico), pois o microserviço recebe 70% das requisições, conforme a configuração do teste.

Na topologia híbrida, mesmo que o microserviço consiga reduzir o processamento no monolítico por responder as requisições, ainda assim, todas as requisições continuam chegando primeiramente ao monolítico, antes de serem desviadas (interceptadas pelo Aspecto) ao microserviço via REST. Esse fato deve ser levado em consideração na leitura do tempo de resposta (Figura 6). Então, embora o tempo de resposta pareça ser o dobro da perspectiva homogênea (de 5k para 10K ms), isso já era esperado pelo fato de que as requisições têm um passo a mais para chegar no microserviço. Sendo assim, ainda é satisfatório pois praticamente manteve a mesma razão em relação ao homogêneo.

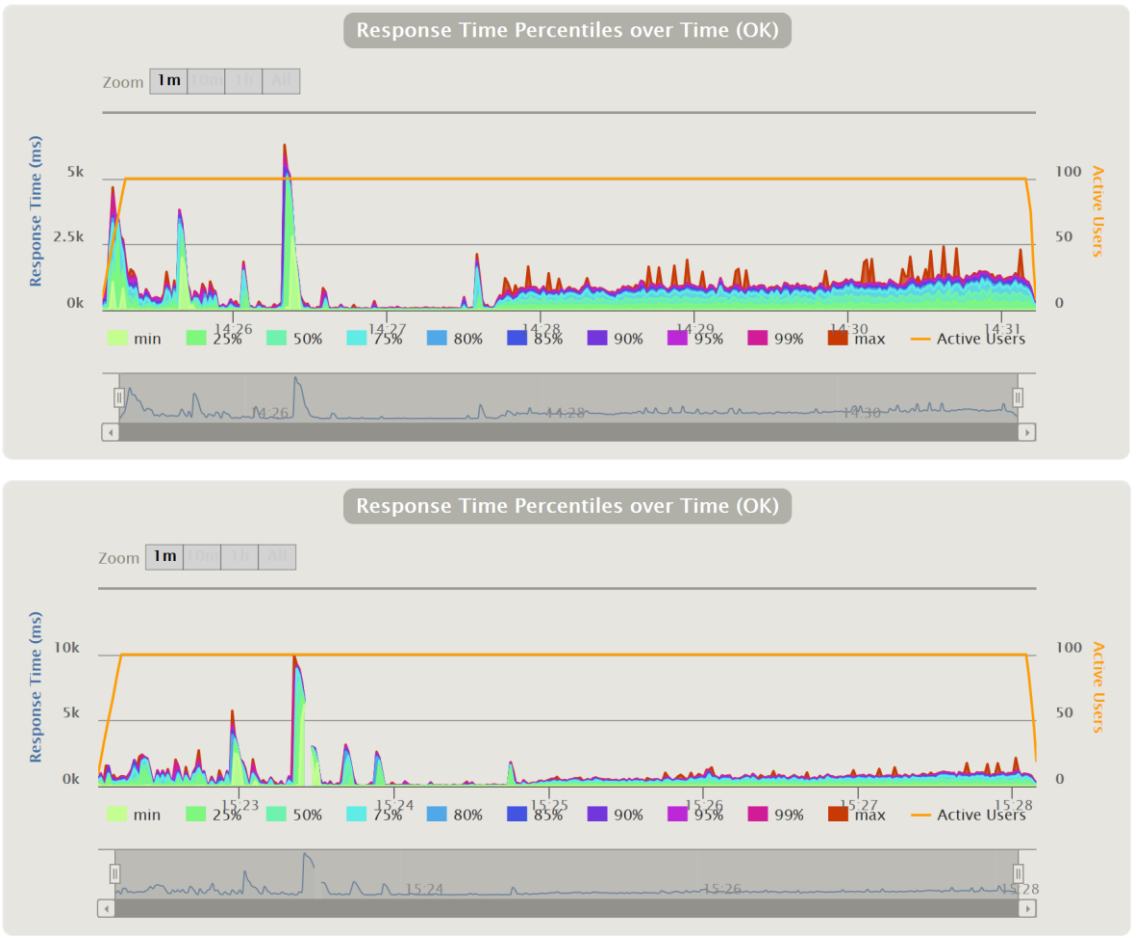

Figure 6. Tempo de resposta: Homogêneo e Híbrido.

4.2. Custo

A avaliação de custo da aplicação segue em duas perspectivas: na Homogênea, que tem apenas uma máquina na nuvem na Amazon, referente ao modelo Monolítico da aplicação; e, na Híbrida, que necessita uma máquina a mais para rodar o microserviço. Pode-se ter uma outra máquina para o serviço de base de dados (Ex: Amazon RDS - Amazon *Relational Database Service*), para ambas perspectivas.

Para a solução proposta, no uso da arquitetura Híbrida, o custo adicional por hora para a máquina do microserviço é de US$ 0,0116 (considerando-se uma máquina de 1 CPU e 1 Gb na Amazon - máquina t2.micro). Se o uso do microserviço for por algumas horas (por exemplo para atender a um determinado pico de demanda de requisições temporário), seu custo seria irrelevante. O uso por um mês completo o dia inteiro traria um custo adicional de aproximadamente 8,352 US$ por mês.

5. Conclusão e Trabalhos Futuros

Migrar todo o monolítico para microserviço ou migrar um módulo ou uma pequena parte do monolítico para microserviço definitivamente não são uma imposição. Certamente grandes organizações tomaram esse caminho. Porém, muitos sistemas tradicionais, nos quais já tem uma cultura legada em produção, podem testar, experimentar ou mesmo migrar temporariamente um de seus serviços para um ambiente

inovador, como o microserviço, apresentando neste trabalho. Um meio de migrar para microserviço sem precisar retirar do ar o monolítico, sem praticamente precisar alterar linhas de código do monolítico e sem ocasionar um *downtime* é o que foi a proposta da abordagem deste trabalho.

Uma aplicação real foi utilizada como prova de conceito para demonstrar que a abordagem proposta permitiu "ir" e "voltar" entre diferentes versões da aplicação sem a necessidade de retirar o sistema do ar tanto no que se refere à modificações de código quanto nos dados da aplicação. Uma avaliação realizada na nuvem demonstra que a abordagem não introduziu *overhead* significativo do ponto de vista de desempenho e nem de custos e ainda pode suportar cargas maiores no monolítico devido à retirada de parte da funcionalidade do monolítico para ser processada em microserviços.

Como trabalho futuro, aplicar um novo *script* de teste sobre dois monolíticos reais em produção (sendo um de empresa e outro de instituição pública) e avaliar o comportamento e desempenho de migração híbrida para microserviço. Também como contribuição futura, replicar a máquina do monolítico e comparar com a replicação das máquinas de microserviço em um número maior na nuvem, na perspectiva híbrida, para avaliar melhor a escalabilidade da aplicação.

Referências

Balalaie, A., Heydarnoori, A., and Jamshidi, P. (2015a). Microservices migration patterns. Technical report, Department of Computer Engineering Sharif University of Technology.

Balalaie, A., Heydarnoori, A., and Jamshidi, P. (2015b). Migrating to cloud-native architectures using microservices: An experience report. In Advances in Service-Oriented and Cloud Computing -Workshops of ESOCC 2015, Taormina, Italy, September 15-17, 2015, Revised Selected Papers, pages 201–215.

Balalaie, A., Heydarnoori, A., and Jamshidi, P. (2016). Microservices architecture enables devops: Migration to a cloud-native architecture. IEEE Software, 33(3):42–52.

Bolar, T. (2018). Integrating microservices with a monolithic application. https://dzone.com/articles/integrating-micro-service-with-monolith-applicatio. Acessado: 18/12/2018.

Eisele, M. Modern Java EE Design Patterns: Building Scalable Architecture for Sustainable Enterprise Development, 2016. O'Reilly Media, Inc.

Fowler, M. and Lewis, J. "Microservices," 25 Mar. 2014; www.martinfowler.com/articles/microservices.html. Acessado: 18/12/2018.

Freire, A. Vídeo de Teste na Nuvem (AWS) da Abordagem de Migração Híbrida com PetClinic. 2019. Disponível em: https://www.dropbox.com/sh/5nzuypfg6xss33r/AAB-iE03-ueXmT2Mb_A1NIv2a?dl=0

Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C. V., Loingtier, J., and Irwin, J. (1997). Aspect-oriented programming. In ECOOP'97 - Object-Oriented Programming, 11th European Conference, Jyv¨askyl¨a, Finland, June 9-13, 1997, Proceedings, pages 220–242.

Levcovitz, A., Terra, R., and Valente, M. T. (2015). Towards a technique for extracting microservices from monolithic enterprise systems. 3rd Brazilian Workshop on Software Visualization, Evolution and Maintenance (VEM), p. 97-104, 2015.

Macero, M. Learn Microservices with Spring Boot, 2017. Apress, Inc.

Medeiros, O., Sampaio, A. and Arraes., A. Uso de AOP na Migração de Aplicações Monolíticas para Microservices. XVI Workshop em Clouds e Aplicações (WCGA), SBRC, 2018.

Newman, S. (2015). Building Microservices Designing Fine-Grained Systems, volume 2015. O'Reilly Media, Inc.

Rey, A. Distributed version of the Spring PetClinic Sample Application built with Spring Cloud. 2016. Disponível em: github.com/spring-petclinic/spring-petclinic-microservices.

Richardson, C. and Floyd, F. (2016). Microservices From Design to Deployment, volume 2016. NGINX, Inc.

Richardson, C. (2014). Pattern: Monolithic architecture. http://microservices.io/patterns/monolithic. Acessado: 18/12/2018.

Richardson, C. (2016b). Refactoring a monolith into microservices. https://www.nginx.com/blog/refactoring-a-monolith-into-microservices/. Acessado: 18/12/2018.

Richardson, C. Microservices Patterns, Manning Publications, 2017.

Syer, D. Spring PetClinic Sample Application. 2015. Disponível em: github.com/spring-projects/spring-petclinic.

Yanaga. E. (2017). Migrating to Microservice Databases from Relational Monolith to Distributed Data, volume 1. O'Reilly