Migração de sistemas monolíticos para microsserviços

Anderson Almeida de Souza Clemente
Centro de Ensino Superior de Juiz de Fora, Juiz de Fora, MG
Evaldo de Oliveira da Silva
Centro de Ensino Superior de Juiz de Fora, Juiz de Fora, MG

Linha de Pesquisa: Trabalho Acadêmico

RESUMO

Este trabalho tem como objetivo apresentar a proposta de um processo de migração de sistemas monolíticos para arquiteturas de microsserviços. O trabalho apresenta a importância do tema dentro das organizações a partir de um levantamento bibliográfico que apresenta a análise dos fatores que influenciam na decisão por este tipo de migração. Existem desafios para realização da migração para microsserviços tornando a arquitetura das aplicações fragmentadas e as vantagens e desvantagens do uso de arquiteturas fragmentadas são apresentadas em um processo de migração. O artigo apresenta um processo baseado em estratégias de migração de sistemas monolíticos para microsserviços e um exemplo de como o processo pode ser implementado.

Palavras-chave: Microsserviços, Arquitetura monolítica, Migração.

1 INTRODUÇÃO

De acordo com Machado (2007), a arquitetura monolítica é ainda a mais utilizada para desenvolvimento no mercado, pois esse tipo de aplicação é feito em um único módulo, que se comunica com o banco de dados. É um tipo de aplicação onde os componentes são interconectados e interdependentes com uma arquitetura de software complexa que prejudica o desenvolvimento de novas funcionalidades, e a manutenção das aplicações (ROUSE, 2016).

Porém, como em outras tecnologias, no desenvolvimento de software também vem ocorrendo evoluções e surgimento de novas tecnologias. Segundo Amaral e

Carvalho (2017) e uma delas é justamente a arquitetura de aplicações web, que pode ser do tipo centralizada, descentralizada ou distribuída. Assim, os autores ressaltam que nem sempre é simples escolher uma arquitetura devido a quantidade de linguagens de programação e banco de dados que podem ser utilizados.

Outro ponto que impacta na escolha da arquitetura da aplicação é que nem sempre o tamanho ou uso da aplicação é o esperado, o que pode resultar em baixa performance, e também o fato de que várias aplicações continuam recebendo atualizações mesmo depois de colocada no ar.

Dessa maneira, um novo conjunto de atividades que apoiam o desenvolvimento vem ganhando destaque, por exemplo, a DevOps, focado em micro serviços (*microsservices*). De acordo com Amaral e Carvalho (2017), o foco dessa arquitetura é a criação de várias micro aplicações independentes de uma aplicação maior, tendo assim cada uma: especificações e responsabilidades dentro da arquitetura.

Assim o desenvolvimento com base em microsserviços permite a criação de várias micro aplicações independentes e ao mesmo tempo com responsabilidades bem definidas para atender o objetivo final da aplicação. A criação de aplicações com base em microsserviços exige o entendimento de uma nova modelagem para a arquitetura das aplicações de software.

Conforme a maturidade da aplicação, ou seja, o estágio de desenvolvimento que se encontra, ela pode continuar sendo monolítica dentro das organizações. A decisão de migrar ou não para uma nova arquitetura depende da necessidade de escalabilidade e de alterações que são realizadas. Fernandes (2020) cita que a abordagem monolítica é viável para aplicações de pequeno porte, e conforme vão crescendo a perda de produtividade é visível, como no caso da Nike e Airbnb. A Airbnb estava demorando cerca de 15 horas de produção semanal devidos a bugs e atualizações no código fonte, e a Nike chegava a demorar 2 dias fazendo o desenvolvimento para passar posteriores 15 dias fazendo a estabilização da aplicação.

Neste contexto, este trabalho propõe um processo de migração de sistemas monolíticos para arquiteturas de microsserviços. O artigo apresenta através de um exemplo como é realizado o processo de migração de um sistema monolítico para um com arquitetura de microsserviços. Para isso, temos os seguintes objetivos

específicos a serem alcançados: definição e entendimento das duas arquiteturas, suas diferenças e particularidades; entendimento e conhecimento das estratégias de migração; e por fim, a apresentação de um estudo de caso.

Como metodologia de pesquisa, utiliza-se de uma pesquisa bibliográfica que visa embasar os conceitos adequados e para que o pesquisador possa se basear em dados já fundamentados, conforme Silva e Menezes (2005). O material escolhido para análise foi artigos publicados em periódicos, bem como livros da área.

O trabalho está dividido da seguinte forma: a Seção 2 apresenta o referencial teórico em que está baseado o trabalho. A Seção 3 descreve as estratégias para migração de sistemas monolíticos para os de arquitetura orientada a serviços. A Seção 4 apresenta a implementação com base no processo proposto. E por fim, a seção 5 faz considerações finais e lista os trabalhos futuros.

2 REFERENCIAL TEÓRICO

Esta seção irá descrever alguns conceitos que fazem parte do tema sobre arquiteturas de software para melhor entender o propósito deste trabalho. Assim buscando esclarecer as motivações que levam ao uso de uma arquitetura fragmentada.

2.1 Sistemas de Monolíticos

Em uma arquitetura monolítica, todas as funcionalidades de um sistema precisam ser implantadas juntas (NEWMAN, 2019), ou seja, todos os serviços que serão utilizados pelo usuário estão no mesmo container. Este modelo faz com que a aplicação independa de outros módulos externos para seu funcionamento.

Há alguns modelos de arquiteturas monolíticas, sendo o mais comum uma aplicação na qual seu código está todo implementado dentro de um único ecossistema. Estes sistemas são compostos por, basicamente, três elementos: *backend*, *frontend* e *database*. Diferente do que muitos pensam, sistemas monólitos também podem ser escalados, tanto verticalmente quanto horizontalmente.

Sistemas monolíticos são facilmente confundidos com sistemas legados. Esses sistemas passaram a ser vistos com um certo preconceito, ou algo a ser evitado, ou ainda algo que remete ao surgimento de problemas (NEWMAN, 2019).

Não obstante, as arquiteturas monolíticas servem para muitos casos de aplicações e negócios, devendo assim ser considerada uma opção.

Os sistemas monolíticos, conforme Richardson (2016) tem como vantagem a maior facilidade no desenvolvimento da aplicação, visto que possuem grande parte das ferramentas convencionais com suporte a essa arquitetura, e também a facilidade da implementação, pois é um único conjunto de código com módulos interdependentes.

Por outro lado, Machado (2017) cita que são aplicações que tendem a se tornar mais complexas com o tempo de desenvolvimento e evolução da mesma; o que pode tornar a manutenção do código um processo complexo e oneroso, principalmente se muitos programadores tiverem trabalhando na sua construção. E por fim, outra desvantagem é que a alteração em um trecho de código pode impactar outros.

2.2 Microsserviços no desenvolvimento de software

O termo programação orientada a serviços foi divulgada em 2002, por Sillitti et al. (2002) em seu livro, onde eram descritos os reflexos e características da mesma.

Temos então que, conforme Bieber e Carpenter (2001), a programação orientada a serviços parte da mesma premissa de que os problemas podem ser modelados em microsserviços relacionados aos objetos que fornecem ou utilizam. Segundo Mardejava (2013), programação orientada a serviços pode também representar passos no processo de um negócio, proporcionando uma entrega econômica de aplicativos de negócios independentes.

Dessa forma, conforme Bieber e Carpenter (2001), um serviço deve ser fornecido de qualquer componente para qualquer um, conforme apresentado na Figura 1, onde vários componentes enviam e recebem mensagens de outros.

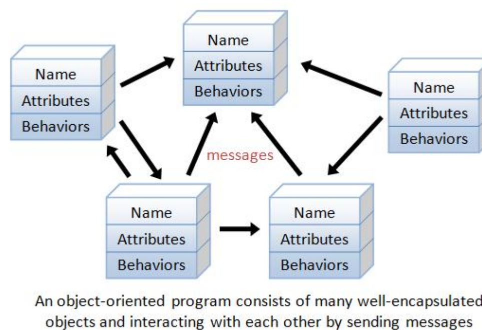


Figura 1: Interação entre vários objetos (Mardejava, 2013).

De acordo com Mardejava (2013), temos os seguintes benefícios do uso dessa tecnologia como o aumento da abstração de criação de novas aplicações, o que aumenta a facilidade de manutenção e inclusão de novas funcionalidades, bem como a unificação das técnicas de desenvolvimento através do uso de um único conceito, para diminuir a complexidade da aplicação, e por fim, essa abordagem permite a redução de custos e a complexidade da solução final.

Cinco componentes que são comuns na arquitetura orientada a serviços. De acordo com Bieber e Carpentes (2001), os componentes são (vide Figura 2): contrato, componente, conector, container e contexto.

O contrato é a onde será definido o comportamento e semântica dos componentes que estão vinculados, sendo então os componentes os elementos computacionais que serão reutilizados por diversas plataformas, ambientes e protocolos. Já os containers são os ambientes onde os componentes são executados, sendo necessário que se faça dentro deles a segurança do código e disponibilidade da aplicação. Por fim, o conector faz o encapsulamento de detalhes de transporte para o contrato e o contexto é o ambiente onde os componentes são de fato implantados, e onde é descrito procedimentos de segurança, instalação, entre outros.

2.3 Arquitetura de Microsserviços

Segundo Machado (2017), o termo microsserviços surgiu na conferência de arquitetura de software (ICSAE) em maio de 2011, desde então bastante difundida na comunidade de engenheiros de software, bem como de desenvolvedores.

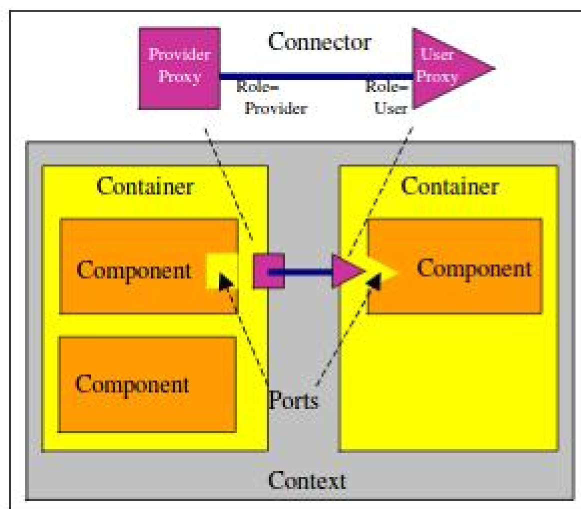


Figura 2: Conexão entre os elementos dos microsserviços (Bieber e Carpentes, 2001).

Algumas das características que definem esse tipo de serviço, conforme Bieber e Carpenter (2001) são:

- Implantabilidade: diz respeito a capacidade que um componente tem de ser reutilizado ou implantando em qualquer ambiente, dessa forma o mesmo deve ser independente de linguagem de programação e plataforma.
- Mobilidade: capacidade de mover o código pela rede, seja através de proxies, interfaces de usuários ou agentes móveis.
- Seguridade: cada serviço deve ter o seu perfil de segurança implementado;
- Conjuntividade: possibilidade de combinar os serviços de uma forma que não tenham sido inicialmente concebidos, o que pode ser feito através de interfaces públicas.
- Disponibilidade: os recursos devem ter alta disponibilidade.

Sabo (2006) também cita como grande vantagem dessa arquitetura a facilidade de escalar a aplicação através do uso da replicação, que espelha servidores de forma a permitir o balanceamento de carga e alta disponibilidade da aplicação. Para fazer o mesmo procedimento na arquitetura monolítica, é necessário

fazer a replicação da aplicação inteira, por isso a que faz uso de microsserviços é mais eficiente.

Kolonay e Sobolewki (2004) também citam algumas dificuldades nesse tipo de arquitetura, tal como o tratamento de manipulação de falhas, tratamento de simultaneidades, tratamento da evolução dos sistemas e uma maior dificuldade no desenvolvimento inicial da aplicação, bem como um custo mais alto.

3. Estratégias para migração de sistemas monolíticos para arquitetura de micro serviços

Apresentamos nessa seção algumas estratégias de migração, como o estrangulador de aplicações (*Strangler Application*), a de separação da aplicação (*backend e frontend*) e extração dos serviços.

Inclusive, um dos procedimentos que deve ser feito após tomada a decisão de fazer a migração é chamado de “Pare de Cavar” (DUARTE JUNIOR, 2018), ou seja, deve-se parar de aumentar a aplicação monolítica e começar o processo de já criar as novas funcionalidades na nova arquitetura, tal como demonstramos na Figura 3.

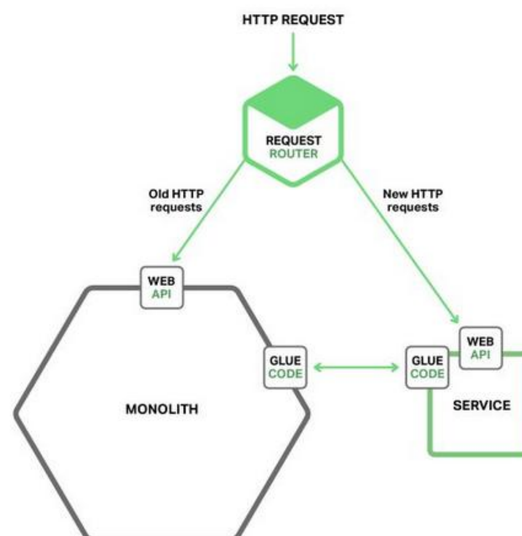


Figura 3: Aplicação do sistema com a nova abordagem (Duarte Junior, 2018).

Após realizar esse passo inicial, temos as estratégias apresentadas na sequência para trabalharem com o código já existente.

3.1. Strangler Application

O padrão do estrangulador, ou *stranger application,* conforme Kawartha (2018) é a incrementação gradual de funcionalidades e serviços específicos, tal como demonstrado na Figura 4. Dessa maneira, conforme os recursos do sistema antigo vão sendo substituídos pelo novo, vai acontecendo o estrangulamento da arquitetura antiga.

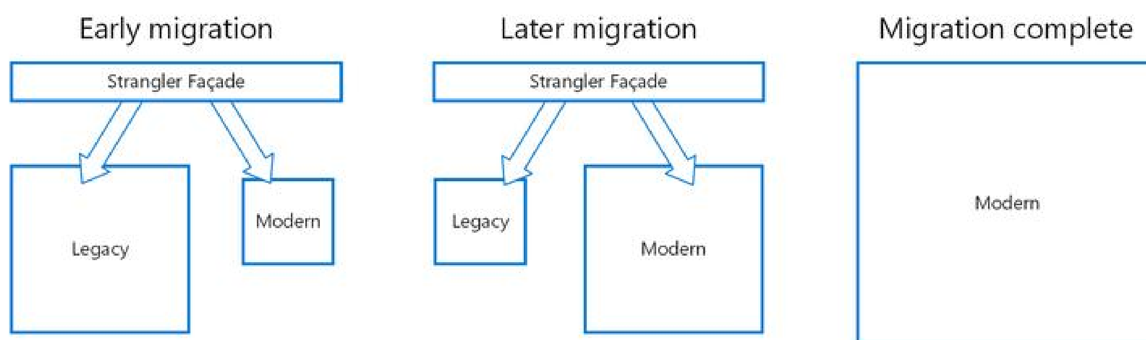


Figura 3: Padrão de estrangulamento (Kawartha, 2018).

Kawartha (2018) cita então que os novos serviços vão sendo colocados à frente da plataforma legada, já no padrão de microsserviços, enquanto os antigos vão sendo refatorados com o tempo. O autor cita ainda que todo esse processo é transparente para o cliente.

Rook (2016) sugere algumas diretrizes sobre como realizar essa migração, de forma a melhor aproveitar o padrão de microsserviços, são elas:

- Adição de um *proxy* entre o aplicativo legado e o usuário;
- Adição de um novo banco de dados vinculado ao *proxy*;
- Implementação da primeira página nesse novo serviço e permissão para que o *proxy* sirva de tráfego para essa página;
- Adição de novas páginas e serviços com o *proxy* configuradas para elas;
- Repetir o processo até que todas as páginas tenham sido migradas;
- E por fim, desligar a aplicação legada.

Segundo Microsoft Padrões (2017), essa forma de migração é adequada para aplicações menores e com menor complexidade e nas aplicações que não serão

desativadas. Assim ambas as arquiteturas podem acessar os recursos enquanto a migração não é concluída.

O ideal é que as aplicações ao serem construídas já prevejam que um dia podem ser migradas para o padrão de microsserviços, sendo já estruturadas inicialmente para que as funcionalidades possam ser interceptadas e migradas, e também devesse atentar para que o *frontend* acompanhe as alterações que estão sendo feitas, conforme Microsoft Padrões (2017), para que não haja perda do desempenho.

Segundo NEWMAN (2019), são definidos três passos para iniciar a implementação do padrão Strangler Application, sendo elas: Em primeiro, qual o serviço é desejado na migração, feito isso, realizar a migração e, por fim, desviar as chamadas do sistema monolítico para a aplicação desenvolvida em microsserviços.

3.2. Estratégia de separação da aplicação em front-end e back-end

Conforme Duarte Junior (2018), essa estratégia tem por objetivo encolher a aplicação monolítica, através da separação da mesma em camadas. Assim, segundo o autor existe pelo menos 3 tipos diferentes de componentes em uma aplicação corporativa, sendo eles: camada de apresentação, camada lógica de negócio e camada de acesso aos dados.

A camada de apresentação é onde estão os componentes que trabalham com as requisições HTTP e a interface de usuário, que quanto mais sofisticada mais código vai ter. A camada lógica de negócios é onde estão implementadas as regras de negócio e comportamento da aplicação, e por fim, a camada de acesso aos dados faz a ligação da camada de negócios com o banco de dados e outros componentes de infraestrutura.

Uma boa prática de programação, de acordo com a W3C é sempre separar a aplicação em camadas, facilitando assim sua manutenção, organização, e caso necessário, sua migração para outra arquitetura. Nesse caso, de acordo com Duarte Junior (2018), a aplicação pode ser separada da seguinte forma demonstrada na Figura 4.

Os benefícios dessa aplicação, conforme Duarte Junior (2018) são o de poder desenvolver, implantar e escalar as aplicações de forma independente e também de poder ser combinada com a estratégia do estrangulador, pois essa estratégia não resolve 100% da migração, apenas parcialmente, e então é necessário uso de outro método.

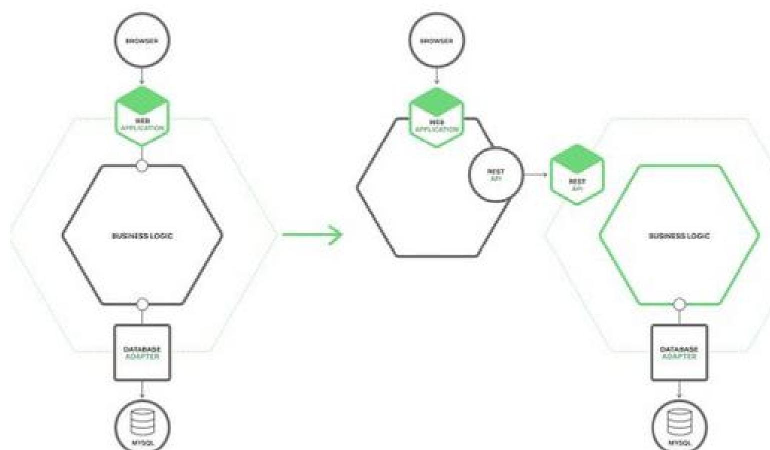


Figura 4: Divisão da aplicação entre back-end e front-end (Duarte Junior, 2018).

3.3. Extração dos serviços

O objetivo dessa estratégia também é de encolher o monolito, segundo Duarte Junior (2018). Para isso são feitas extrações de módulos independentes, que são transformados em serviços, encolhendo assim a outra arquitetura, assim, conforme mais módulos forem convertidos ou o monolítico deixará de existir ou vai se tornar um serviço por si só, segundo o autor.

Esse processo é demonstrado na Figura 5, onde o primeiro passo é definir a interface entre o módulo e o monolítico, considerando que as duas arquiteturas vão precisar continuar tendo acesso a base de dados. Caso a arquitetura legada tenha muitas associações entre as classes será uma tarefa mais complexa, podendo exigir uma refatoração no código antes de iniciar a migração, segundo Duarte Junior (2018).

No exemplo da Figura 5, o candidato a ser extraído é o módulo X, onde seus componentes estão sendo utilizados por X e Y. Temos assim os seguintes passos a serem realizados:

- Refatoração de alto nível com a interface de entrada sendo o módulo X a invocar o módulo Z, e então uma interface de saída usada pelo módulo Z e invocando o Y;
- Transformar o módulo em um serviço separado já que ele já foi isolado;

- Refazer as etapas de refatorar e isolar os serviços até que todos os módulos tenham passado pelo processo;
- Por fim, o monólito não será mais necessário e poderá ser desligado.

O desafio nesse tipo de migração é que nas aplicações muito grandes existe o desafio da escolha de por quais módulos começar a migração, pois todos são candidatos, nesse caso Duarte Junior (2018) indica começar pelos módulos mais fáceis de extrair, e posteriormente escolher os que tem alterações com maior frequência.

Uma outra indicação do autor é também a de fazer a extração de módulos que possuem características bem divergentes do restante da aplicação, assim partes do código que consumem muitos recursos como memória e ou processamento já podem ser isolados em micro serviços e escalados.

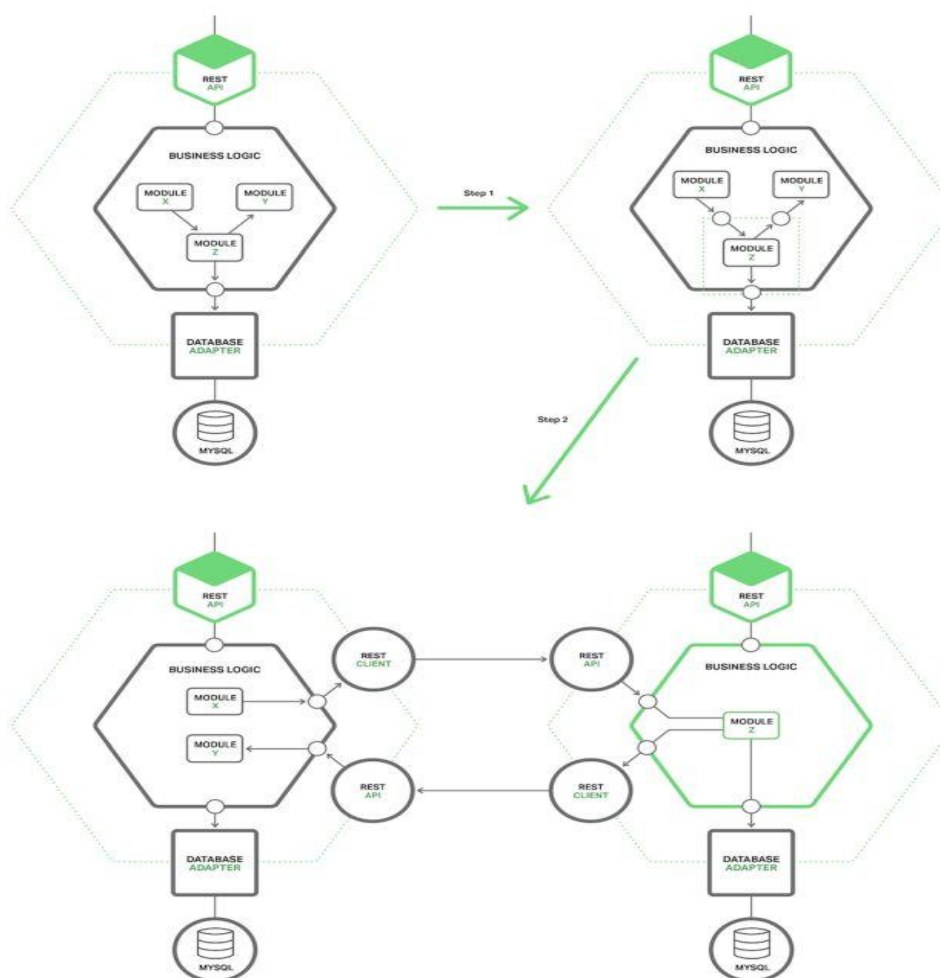


Figura 5: Diagrama de extração de módulos para a arquitetura de micro serviços (Duarte Junior, 2018).

4. O uso da estratégia *Strangler Application* para migração de sistemas monolíticos para arquitetura de microsserviços

De maneira a demonstrar um processo de migração, utilizaremos como base um sistema de biblioteca construído em arquitetura monolítica. A solução de migração é um protótipo que utiliza a linguagem Java e seus frameworks como plataforma de desenvolvimento e, nessa seção serão apresentados os aspectos de desenvolvimento da extração de um serviço do monolítico para microsserviços. Foram utilizados os três passos definidos por Sam Newman e citado na seção 3.1 deste trabalho. Para garantir o uso do padrão do estrangulador, os demais serviços deverão continuar disponíveis na aplicação monolítica e, desta forma, inicia-se o processo de migração da aplicação.

4.1. Implementação da estratégia *Strangler Application*

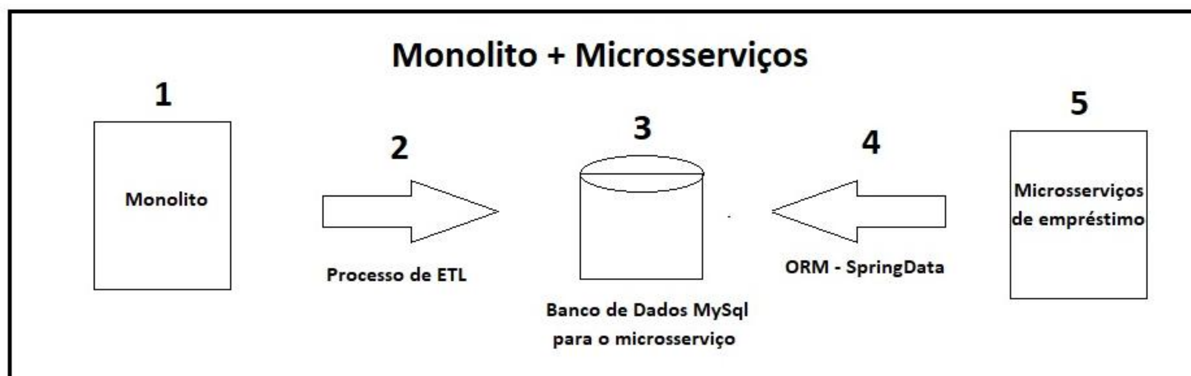
A escolha do padrão ou estratégia *strangler application* – estrangulamento da aplicação - é motivada pela necessidade de disponibilidade da aplicação e seus serviços enquanto ocorre a migração. Esse padrão elimina a necessidade de fazer alterações no sistema existente, além disso, por ser uma migração incremental e de uma pequena fatia do monólito, torna-se mais fácil pausar ou interromper um desenvolvimento e cada etapa é altamente reversível, reduzindo os riscos e os custos.

O estudo de caso é baseado em uma aplicação monolítica que contém inúmeros serviços. Inicialmente será escolhido um serviço a partir da necessidade de migração, além de outros fatores que ajudam a definir qual serviço deverá ser migrado primeiro. Enquanto isso, os outros serviços deverão continuar disponíveis.

Para definir qual o primeiro serviço a ser migrado, alguns pontos foram considerados, por exemplo: a complexidade, o domínio e a necessidade. Desta forma, foi decidido migrar o serviço de empréstimo de livros. É um serviço cujo domínio é simples, de baixa complexidade e pode haver a necessidade de ser escalável. Um exemplo prático de um serviço que poderia ser migrado, porém fere alguns conceitos da decisão é o serviço de cadastro de livros. Este, por sua vez, possui um domínio consideravelmente mais complexo e, como não é um serviço

usado sempre, apenas quando há a necessidade de um novo cadastro, entende-se que há pequenas chances da necessidade de ser escalável.

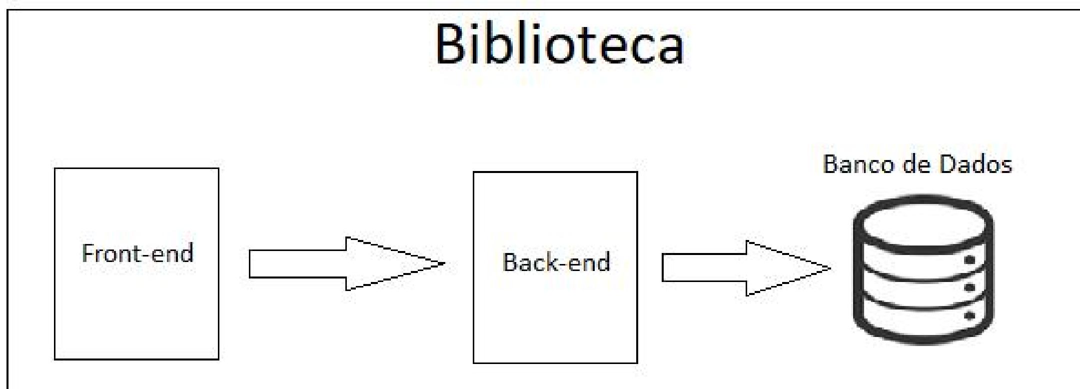
Decidido o serviço migrado, foi realizado um desenho de como as aplicações deverão se comunicar.



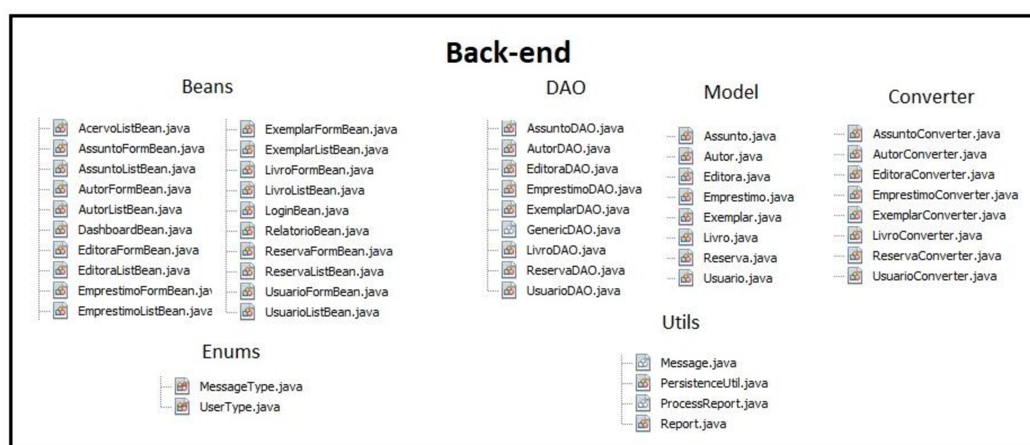
Baseado na imagem, o fluxo segue da seguinte forma: Na primeira etapa teremos o sistema monolítico (1) com todos os serviços e uma única base. Para que o microsserviço de empréstimo tenha acesso aos dados necessários para seu funcionamento, foi criado um processo de extração, tratamento e carregamento dos dados (ETL) (2). Para o estudo de caso, por ser um modelo simples, com poucos dados, a extração ocorre manualmente por arquivo csv. Os dados são carregados na base do microsserviço (3) e ficam disponíveis para serem acessados pelo serviço de empréstimo (5) por qualquer ORM (4). Enquanto o serviço de cadastro de livros não for migrado, a atualização do banco de dados (3) com as informações do sistema antigo, deverão acontecer diariamente como forma de atualização. Uma sugestão para aplicações maiores é automatizar o processo de ETL, utilizando, por exemplo o universo Apache Spark (scala ou Python). Desta forma, quando uma atualização é feita em um registro no legado, o processo de ETL automaticamente será executado e atualizará a base do microsserviço.

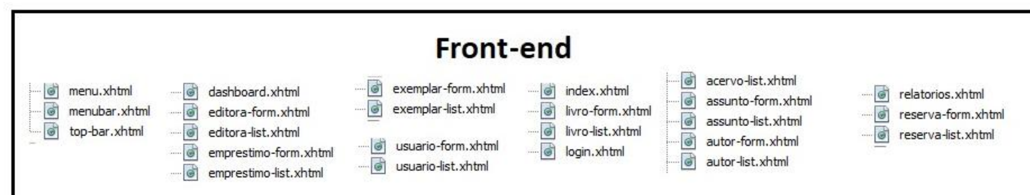
4.2. Estudo de caso para a migração do sistema de Biblioteca Acadêmica

Abaixo segue um desenho da aplicação monolítica apresentando sua arquitetura.



A imagem a seguir apresenta detalhes do sistema monolítico:





O processo de migração, inicia-se com a criação de alguns componentes de infraestrutura que auxiliam os microsserviços. Para isso utilizaremos Spring Cloud, cujo objetivo principal é fornecer uma integração entre o Spring Boot e o projeto Netflix OSS. Serão criadas três APIs auxiliadoras que são: API Zuul gateway, Server Discovery e uma API de autenticação.

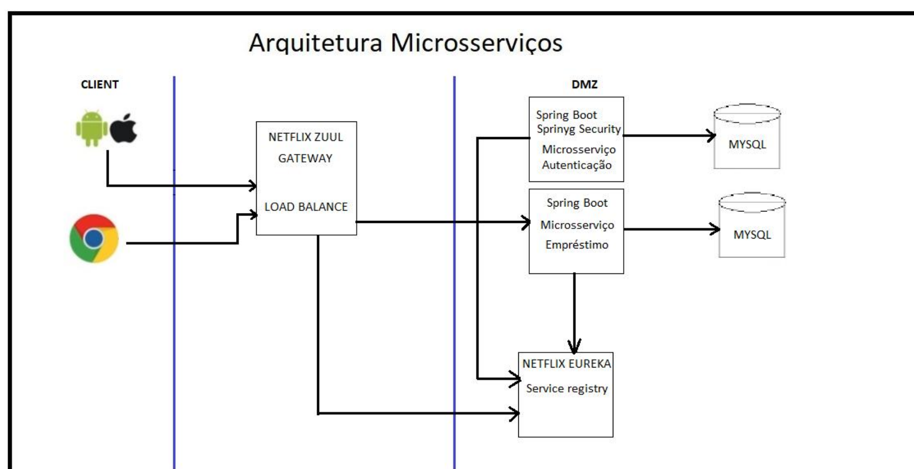
O objetivo da API gateway é gerenciar os microsserviços. Além disso, a API gateway protege, monitora e analisa as APIs de uma forma centralizada. Todas as requisições aos microsserviços deverão passar por essa API e então redirecionadas para o serviços chamados. Esse gateway também é responsável por fazer o

balanceamento de carga – *load balance* – da aplicação, quando mais de uma instância for registrada no *Service Discovery*.

Um Service Discovery foi criado, utilizando Eureka, um projeto *open source* da Netflix. O objetivo dessa API é registrar as aplicações de forma centralizada, deixando-a altamente disponível devidas as replicações. O Eureka deve ser usado no server e no cliente.

Foi criado um microsserviço de autenticação utilizando Spring Security que será responsável por autenticar e autorizar as requisições nos microsserviços. A premissa para este serviço funcionar é ter uma base com os usuários cadastrados no sistema monolítico. Não está dentro do escopo do trabalho apresentar como será feito o processo de migração dos dados, para este caso, a base foi extraída em CSV e importada na base do novo sistema. Quando o usuário faz o *login,* a API de autenticação retorna um token e este deverá ser passado no header das requisições para acessar os serviços. Todos os microsserviços deverão estar com a implementação do Spring Security e JWT para controlar o acesso.

Desta forma, temos a seguinte estrutura para os microsserviços:



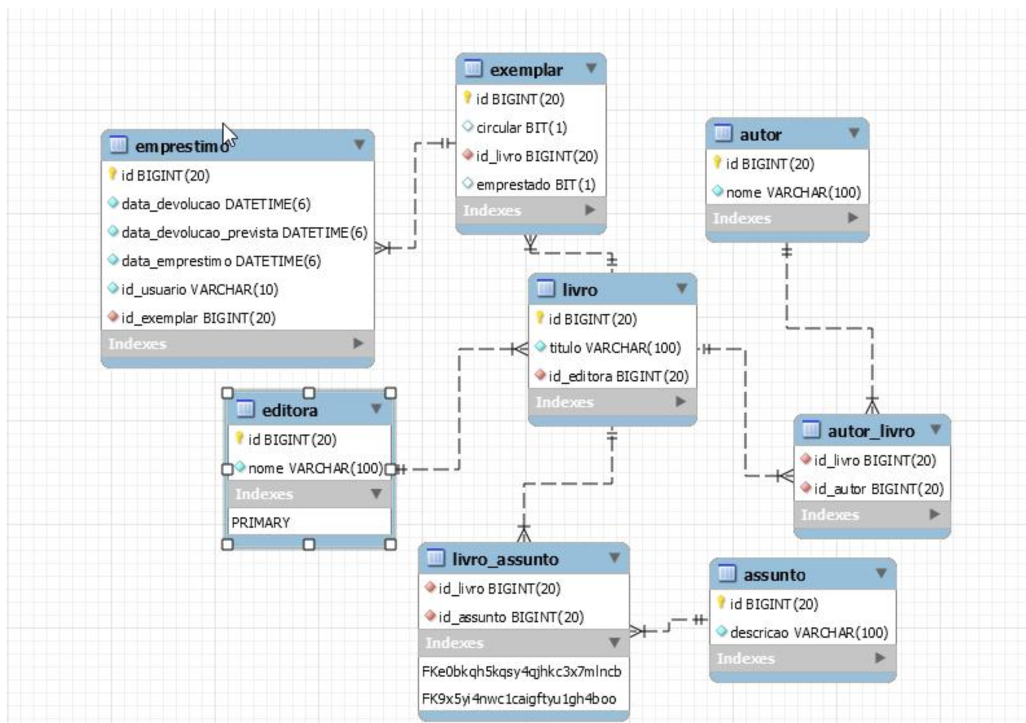
No projeto, os componentes estão representados da seguinte forma:

auth [boot] [devtools]
discovery [boot] [devtools]
emprestimo [boot] [devtools]
gateway [boot] [devtools]

Para a migração do serviço de empréstimo, foi necessário extrair as informações de livros, autores, editores, assuntos e exemplares do monolítico, por csv e incluída no banco de dados do microsserviço. Esses dados são exibidos no

serviço de empréstimo em forma de lista para apresentar as possibilidades de empréstimos.

O modelo do microsserviço ficou da seguinte forma:



Para que não haja alterações no monolito, a carga para atualizar as tabelas do microsserviço pode ser realizada todos os dias de forma automatizada. Neste caso deve-se criar uma rotina que realiza as cargas históricas e incrementais, respeitando inserção e atualização.

Todas as requisições deverão passar pela API gateway, que está sendo registrada na porta 8080, essa API, por sua vez, fará o roteamento para o microsserviço correspondente. Segue um exemplo de uma chamada via p*ostman* para o serviço de login:

POST http://localhost:8080/api/auth/login

Send | Save

Params | Authorization | Headers (9) | Body | Pre-request Script | Tests | Settings | Cookies

none | form-data | x-www-form-urlencoded | raw | binary | GraphQL | JSON


```
1 {
2     "userName": "admin",
3     "password": "123456"
4 }
```


Body | Cookies | Headers (11) | Test Results — Status: 200 OK Time: 1439 ms Size: 532 B Save Respon

Pretty | Raw | Preview | Visualize | JSON


```
1 {
2     "userName": "admin",
3     "token": "eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJhZG1pbiIsInJvbGVzIjpbIkFkbWluIl0sImlhdCI6MTYwNTI4ODI2OCwiZXhwIjoxNjA1Mjg4MzA0fQ.LWf-5aDnhFvJrHx
4 }
```


Na imagem abaixo, é possível ver as aplicações registradas no Service Discovery e, para apresentar a escalabilidade das aplicações, serão registradas duas instâncias do microsserviço de empréstimo. A API gateway fará o balanceamento de acordo com a demanda. Cada vez que o serviço for chamado, será redirecionado para uma instância disponível.

Application	AMIs	Availability Zones	Status
AUTH	n/a (1)	(1)	UP (1) - localhost:auth:8083
EMPRESTIMO	n/a (2)	(2)	UP (2) - localhost:emprestimo:9091 , localhost:emprestimo:9090
GATEWAY	n/a (1)	(1)	UP (1) - localhost:gateway:8080

O microsserviço de empréstimo está disponível nas portas 9090 e 9091.

Desta forma temos uma arquitetura que é capaz de disponibilizar e gerenciar inúmeros microsserviços.

4.3. Vantagens da migração de sistemas monolíticos para arquitetura de microsserviços

A decisão de migrar uma aplicação monolítica para uma arquitetura de microsserviços deve estar interligada às vantagens deste processo. Baseado no estudo de caso, após a migração do serviço de empréstimo de livros, o módulo se tornou mais fácil de desenvolver e manter, deploy acontece de forma mais rápida, alterações podem ser facilmente implementadas, possibilitou escalar o serviço de forma independente, melhorou o isolamento de falhas e aumento significativo de disponibilidade

5 Considerações Finais

Antes de um sistema entrar de fato em produção, é complexa a tarefa de medir qual será sua necessidade de escalabilidade, interoperabilidade e de revisões e manutenções periódicas. Esse fato influencia na arquitetura e na forma em que o sistema será desenvolvido e, devido a facilidade, a maior parte das aplicações são feitas inicialmente na arquitetura monolítica.

Com o sistema já em produção, pode haver então a necessidade de fazer sua migração para uma arquitetura mais robusta e que trazem benefícios que não podem ser alcançados com monólitos.

Algumas das principais formas de migração conhecidas são o estrangulador, separação da mesma em *back-end* e *front-end* e por extração de serviços. Cada uma tem seus pontos positivos e negativos, e em linhas gerais a ideia de todas é que seja realizada uma migração periódica até que todos os módulos do sistema sejam migrados para a arquitetura de microsserviços.

Apresentamos então um estudo de caso migrando um serviço do sistema monolítico da biblioteca, utilizando o padrão do estrangulador de aplicação, com base na reengenharia de software, desenvolvendo uma nova versão melhorada e projetada para arquitetura de microsserviços.

Após a finalização da migração, o serviço de empréstimo de livros obteve algumas melhorias como, por exemplo: maior facilidade de manutenção, disponibilidade enquanto os outros serviços estiverem passando por manutenção ou outra espécie de indisponibilidade e maior capacidade de escalabilidade.

REFERÊNCIAS

AMARAL, Odravison & CARVALHO, Marcus. **Arquitetura de micro serviços: uma comparação com sistemas monolíticos.** UFPB. 2017.

BIEBER, Guy & CARPENTES, Jeff. **Introduction to Service-Oriented Programming**. 2001. Disponível em <https://pdfs.semanticscholar.org/988e/b06972d3dc1335f7b266883b853af7ccbd20.pdf.>. Acessado em 15 de junho de 2020.

DUARTE JUNIOR, Luiz F. **Dicas para refatorar um monolito em microsserviços.** 2018. Disponível em <https://imasters.com.br/apis-microsservicos/dicas-para-refatorar-um-monolito-em-microsservicos>. Acessado em 28 de setembro de 2020.

FERNANDES, Igor. **Um Estudo de Caso — Migração para Microsserviços: Airbnb & Nike.** 2020. Disponível em <https://medium.com/@ifc_7168/um-estudo-de-caso-migra%C3%A7%C3%A3o-para-microsservi%C3%A7os-airbnb-nike-d6cc1df69df8>. Acessado em 25 de setembro de 2020.

KAWARTHA, Piotr. **Progressive Web App strategies — Strangler Pattern.** 2018. Disponível em <https://medium.com/the-vue-storefront-journal/progressive-web-app-strategies-strangler-pattern-493ce61d4641>. Acessado em 30 de setembro de 2020.

KOLONAY, R. M. & SOBOLEWSKI, M. ***Grid interactive service-oriented programming environment.*** *2004. Disponível em <http://w.sorcersoft.org/publications/papers/CE2004-557.pdf>. Acessado em 18 de junho de 2020.*

MACHADO, M. G. ***Micro Serviços: Qual a diferença para arquitetura monolítica?*** *2007. Disponível em <http://www.opus-software.com.br/microservicos-diferenca-arquietura-monoliticas/>. Acessado em 24 de setembro de 2020.*

MARDEJAVA. Disponível em <http://mahderjava.blogspot.com/2013/04/is-service-oriented-programming-sop.html>. 2013. Acessado em 18 de junho de 2020.

MICROSOFT PADRÕES. **Stranger Pattern.** 2017. Disponível em <https://docs.microsoft.com/en-us/azure/architecture/patterns/strangler>. Acessado em 30 de setembro de 2020.

RICHARDSON, C. **Pattern: Microservices Architecture.** 2016. Disponível em < http://microservices.io/patterns/microservices.html >. Acessado em 24 de setembro de 2020.

ROOK. Michiel. **The stranger pattern in practice.** 2016. Disponível em <https://www.michielrook.nl/2016/11/strangler-pattern-practice/>. Acessado em 30 de setembro de 2020.

ROUSE, M. **Monolithic Architecture.** 2016. Disponível em <http://whatis.techtarget.com/definition/monolithic-architecture>. Acessado em 01 de setembro de 2020.

SABO, C. P. **Avaliação de Desempenho com Algoritmos de Escalonamento em Clusters de Servidores Web** 2006, São Carlos, cap. 3, p. 35.

SILLITTI, A. & VERNAZZA, T. & SUCCI, G. **Service oriented programming: a new paradigm of *software reuse***, in 7th International Conference on *Software Reuse. Springer, Berlim. 2002.*

SILVA, Edna Lúcia da; MENEZES, Estera Muszkat. **Metodologia da Pesquisa e Elaboração de Dissertação**. 2005. 4a ed. Florianópolis: Universidade Federal de Santa Catarina – UFSC. Disponível em <https://projetos.inf.ufsc.br/arquivos/Metodologia_de_pesquisa_e_elaboracao_de_teses_e_dissertacoes_4ed.pdf>. Acessado em 21 setembro de 2020.

W3C. **W3C Brasil**. 2020. Disponível em <https://www.w3c.br/Padroes/>. Acessado em 27 de setembro de 2020.

NEWMAN, Sam. **Migrando sistemas monolíticos para microsserviços**. 2020. " O'Reilly Media, Inc.".