Pontifícia Universidade Católica de São Paulo

Coordenadoria Geral de Especialização,
Aperfeiçoamento e Extensão

Guilherme Jorge Aragão da Cruz

A Arquitetura de Microsserviços em Empresas: Estudo de Caso da sua Aplicabilidade em um Processo de Negócio

Especialista Em Engenharia De Software

São Paulo

2020

Pontifícia Universidade Católica de São Paulo

Guilherme Jorge Aragão da Cruz

A Arquitetura de Microsserviços em Empresas: Estudo de Caso da sua Aplicabilidade em um Processo de Negócio

Monografia apresentada à Banca Examinadora da Pontifícia Universidade Católica de São Paulo, como exigência parcial para obtenção do título de ESPECIALISTA em Engenharia de Software.

Orientador: Prof. Dr. André Luiz Garcia Pereira.

São Paulo

2020

Agradecimentos

Aos meus amigos, amigas e familiares pela paciência, apoio e compreensão nos mais diversos momentos; bem como aos amigos de curso por todo apoio e tempo oferecido.

Aos amigos de trabalho pelo apoio e acompanhamento de toda minha trajetória acadêmica ao longo destes anos.

Ao meu orientador, professor André Pereira, pela aceitação do tema proposto e por todo apoio e ajuda nesta e em outras disciplinas.

Glossário

API: *Application Programming Interface* - conjunto de definições e protocolos usados no desenvolvimento e na integração de software.

ESB: *Enterprise Service Bus* - o barramento de serviços, que permite acoplar diferentes tecnologias.

FTP: *File Transfer Protocol* - protocolo genérico independente de hardware para transferir arquivos entre recursos computacionais.

HATEOAS: *Hypermedia As the Engine Of Application State* - componente da arquitetura REST para navegação e descoberta dinâmica.

HTTP: *Hypertext Transfer Protocol* - protocolo de comunicação utilizado para sistemas de informação de hipermídia, distribuídos e colaborativos.

JSON: *JavaScript Object Notation* - formato compacto, de padrão aberto independente, de troca de dados simples e rápida entre sistemas.

REST: *Representational State Transfer* - estilo de arquitetura de software que define um conjunto de restrições a serem usadas para a criação de serviços.

RMM: *Richardson Maturity Model* - modelo de referência de boas práticas para criação de APIs de qualidade.

ROI: *Return on Investment* - a relação entre o dinheiro ganho como resultado de um eventual investimento.

RPC: *Remote Procedure Call* - tecnologia de comunicação entre processos que permite a um programa de computador chamar um procedimento.

SGBD: Sistema de Gerenciamento de Banco de Dados - conjunto responsável pelo gerenciamento de um banco de dados.

SOA: *Service-Oriented Architecture* - modelo de arquitetura de sistemas que prega que suas funcionalidades sejam expostas como serviços.

SOAP: *Simple Object Access Protocol* - padrão de protocolo para troca de informações.

UML: Unified Modeling Language - linguagem padrão para a elaboração da estrutura de projetos de software.

WS: *Web Services* - tecnologia que permite a comunicação entre aplicações de diferentes linguagens e sistema operacional.

WSDL: *Web Services Description Language* - linguagem baseada em XML para a descrição de *webservices*.

XML: *eXtensible Markup Language* - linguagem para a marcação de dados para necessidades especiais.

Resumo

A Arquitetura de Microsserviços ou *Microservices Architecture* tem se tornado a cada dia um assunto mais presente nas organizações, que exigem cada vez mais o emprego de soluções de software de alta eficiência aliada à velocidade de desenvolvimento. Partindo desse contexto, o projeto tem por objetivo apresentar os principais conceitos relativos à arquitetura de microsserviços, incluindo aspectos de seu nascimento, seus paradigmas, suas aplicações e seus comparativos com outras abordagens. Também é objetivo deste trabalho apresentar um estudo de caso com implementação real de microsserviços em um sistema de informação voltado a um processo de negócio, que é o objeto de estudo do projeto.

Palavras chaves: Arquitetura de microsserviços, serviços web, sistemas de informação, processos de negócio.

Abstract

Microservices Architecture has become an increasingly common issue in organizations, which increasingly demand the use of highly efficient software solutions coupled with the speed of development. From this context, the project aims to present the main concepts related to microservices architecture, including aspects of its birth, its paradigms, its applications and its comparisons with other approaches. It is also the objective of this work to present a case study with real implementation of microservices in an information system focused on a business process, which is the object of study of the project.

Keywords: Microservices architecture, web services, information systems, business processes.

Lista de Figuras

Lista de Tabelas

Sumário

1 Introdução

1.1 Motivação

As últimas décadas apresentaram forte impacto e tiveram um importante papel em relação a novos desafios de processos inerentes às organizações, em que processos de negócio passam por constantes evoluções e os sistemas que realizam a sua sustentação precisam acompanhar este crescimento, criando assim, também, um cenário de desafio tecnológico. Historicamente, arquiteturas de aplicações fortemente acopladas criam um paradoxo frente a esse desafio, em que a tecnologia infere diretamente nos indicadores de negócios; havendo assim, a necessidade de estruturar a arquitetura de aplicações de uma forma voltada não somente às tecnologias, mas também frente à própria finalidade de negócio das organizações.

De acordo com a *Workflow Management Coalition* (1999), "um processo de negócio consiste em um conjunto de atividades relacionadas que visam atingir um único objetivo de negócio, no contexto de uma estrutura organizacional, definindo regras e relacionamentos." (p. 7). Ainda, segundo DUMAS, Marlon et al (2013), um "processo de negócio", ou *business process* são atividades críticas para o sucesso das organizações. Desta forma, o desenho funcional de um processo de negócio deve ser bem estruturado, devido a sua grande importância para a eficiência dos produtos e serviços das organizações; porém a sua organização arquitetural também deve ser levada em conta.

Processos de negócio têm sofrido constantes transformações nas organizações, tornando-se cada vez mais um item com características de grande dinamismo e de evolução constante; são um dos principais pilares para o diferencial competitivo na atualidade. Segundo Cruz (2010), “negócio é a reunião de três elementos: pessoas, processos e Tecnologia da Informação, com a finalidade de atender as necessidades do cliente” (p. 46). Desta forma há uma importância central nesses tópicos, que devem ser sempre levados em consideração pelas organizações.

Sistemas de informação que sustentam processos críticos de negócio são componentes fundamentais das organizações e requisitos não funcionais como interoperabilidade, tolerância a falhas, escalabilidade, recuperabilidade (também

conhecida como “resiliência”) e modificabilidade (ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS, 2003) são cruciais para a garantia de continuidade do negócio.

Frente a essas características, é importante que as organizações possuam um nível adequado de maturidade em suas aplicações, a fim de facilitar o entendimento e desenvolvimento de novas interfaces e aplicações; surgindo assim a necessidade de definição de modelos de maturidade que possibilitem a organização de uma estratégia de evolução em seus sistemas (FOWLER, 2010).

1.2 Objetivo Geral

O presente trabalho teve como objetivo macro apresentar uma proposta do emprego da arquitetura de microsserviços em um sistema de informação, com base em um estudo de caso de uma empresa do segmento de meios de pagamentos.

Para ilustração e melhor entendimento do estudo, foi elaborada uma proposta de arquitetura de microsserviços baseada no produto do objeto de estudo de caso, com foco em um de seus requisitos funcionais de negócio (cancelamento de uma venda).

1.3 Objetivos Específicos

Os objetivos específicos deste trabalho foram:

- Levantar e descrever o contexto atual do cenário de microsserviços: suas principais características, abordagem dos conceitos tecnológicos de API, JSON, REST e *RESTful* e o Modelo de Maturidade da Arquitetura de APIs;
- Traçar um comparativo entre microsserviços e o SOA tradicional: elencar os principais pontos de convergência das abordagens e esclarecer possíveis mitos;

- Entender e descrever o cenário da empresa que motivou a adoção de uma aplicação baseada na arquitetura de microsserviços;
- Apresentar e explicar a arquitetura elaborada para o problema, suas vantagens e desvantagens.

1.4 Delimitação do Problema

Além das definições conceituais da arquitetura de microsserviços, apresentadas na revisão bibliográfica deste trabalho, face às diversas possibilidades que o tema implica no universo de processos de negócio, o presente estudo foi voltado, essencialmente, à sua aplicabilidade dentro do projeto objeto de estudo; tendo suas principais vantagens – e eventuais desvantagens – elencadas de forma direta ao próprio caso. Deste modo, a ênfase da pesquisa está, de forma frequente, diretamente relacionada ao estudo de caso.

1.5 Justificativa

Diante do cenário exposto no capítulo 1.1, verifica-se a necessidade das empresas manterem ou construírem e evoluírem sistemas de informação que possam manter as regras de negócio sob seu próprio domínio e que, ainda assim, atendam aos requisitos não funcionais já citados, facilitando também integrações com parceiros e colaboradores diretos.

A pesquisa realizada também elenca as principais vantagens da abordagem da arquitetura de microsserviços, um modelo de avaliação da sua maturidade, bem como suas possíveis desvantagens para diferentes cenários de uso.

2 Revisão Bibliográfica

Este capítulo tem como objetivo apresentar microsserviços e seus principais conceitos e posicionamento no cenário atual, abordando suas principais características bem como um modelo de maturidade aderente ao estudo de caso deste trabalho.

Por fim, também será apresentado um breve comparativo entre microsserviços e a abordagem de Arquitetura Orientada a Serviços ou *Service-Oriented Architecture* (conhecida principalmente por sua sigla, “SOA”).

2.1 Visão Geral

Atualmente, a arquitetura de microsserviços tem se destacado cada vez mais como um padrão de estilo arquitetural para o desenvolvimento de aplicações corporativas. É importante contextualizar aqui a sua definição: a arquitetura de microsserviços é uma abordagem que entrega uma única aplicação por meio de um conjunto de pequenos serviços (FOWER, LEWIS 2014). Serviços devem empregar mecanismos leves para sua comunicação (muitas vezes por meio de uma *Application Programming Interface* – API com recursos do *HiperText Transfer Protocol – HTTP*), permitindo assim um modelo de comunicação amplo e padronizado (ERL, 2009).

Espera-se também que esses serviços pertençam a pequenas equipes autossuficientes, evitando um gerenciamento centralizado, com independência de linguagens de programação e tecnologia de armazenamento de dados. (JUNG, Matthias et al, 2016).

Além dos serviços já mencionados, alguns outros componentes aparecem em uma arquitetura de microsserviços típica:

- ***Gateway* de API:** Ponto de entrada para os clientes. Com o emprego de um *gateway* de API, a invocação de um serviço não ocorre diretamente, uma vez que, recebida a requisição, há o encaminhamento para os serviços adequados no *backend*. Também trata a camada de autenticação e autorização de forma centralizada, protegendo os serviços.

- **Gerenciamento/Orquestração:** Componente responsável pela disposição dos serviços em nodos, possibilita identificar falhas e trabalhar o balanceamento de carga entre serviços.
- **Armazenamento de Dados:** Responsável pelos mecanismos de persistência, como um banco de dados, arquivos de log, entre outros. O armazenamento de dados é separado para cada microsserviço.

A Figura 1 representa um modelo de interação entre esses componentes:

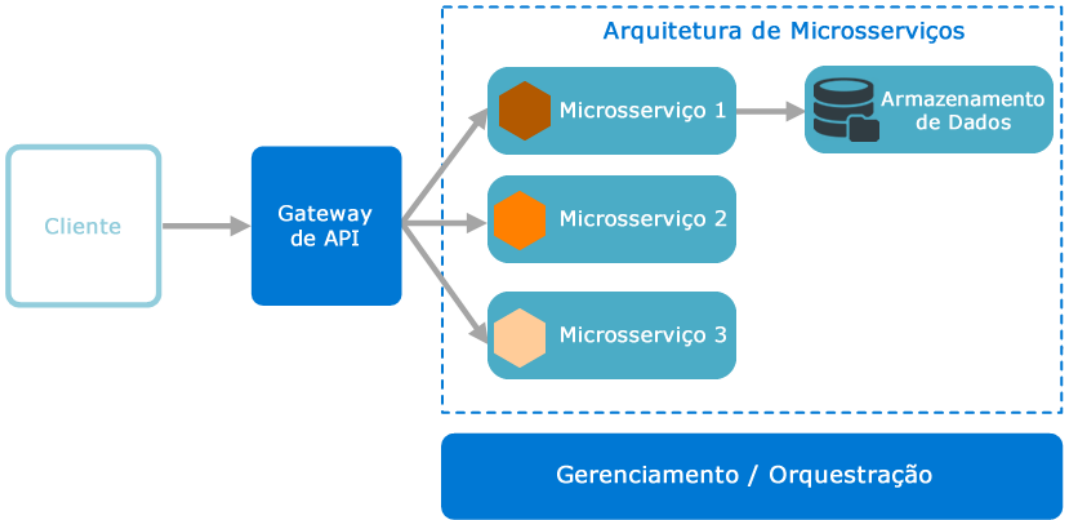

Figura 1 - Modelo de interação de componentes da arquitetura de microsserviços
Fonte: Adaptado de Microsoft (2019)

A seguir serão abordadas as principais características inerentes à arquitetura de microsserviços, bem como demais definições de termos que serão utilizados frequentemente ao longo do trabalho.

2.1.1 Principais Características

Em linhas gerais, a literatura especializada elenca diversas características inerentes a aplicações baseadas em microsserviços. Não há, contudo, uma quantidade exata (FOWER, LEWIS 2014). Para o presente trabalho, serão abordadas as características que foram fundamentais para a tomada de decisão ao se adotar uma arquitetura de microsserviços para o projeto da empresa, objeto do estudo de caso.

2.1.1.1 Interoperabilidade

A interoperabilidade, dentro do contexto de microsserviços, consiste na capacidade que um serviço possui de se comunicar com outros serviços – inclusive entre tecnologias distintas, como uma integração Java *versus* Python, por exemplo – de maneira transparente (JOSUTTIS, 2008).

A comunicação entre os microsserviços é inspirada no estilo Unix Clássico de recebimento da requisição, do processamento e da resposta, e pode utilizar comunicações síncronas ou assíncronas. A figura 2 ilustra como ocorre a interoperabilidade de comunicação em uma arquitetura de microsserviços.

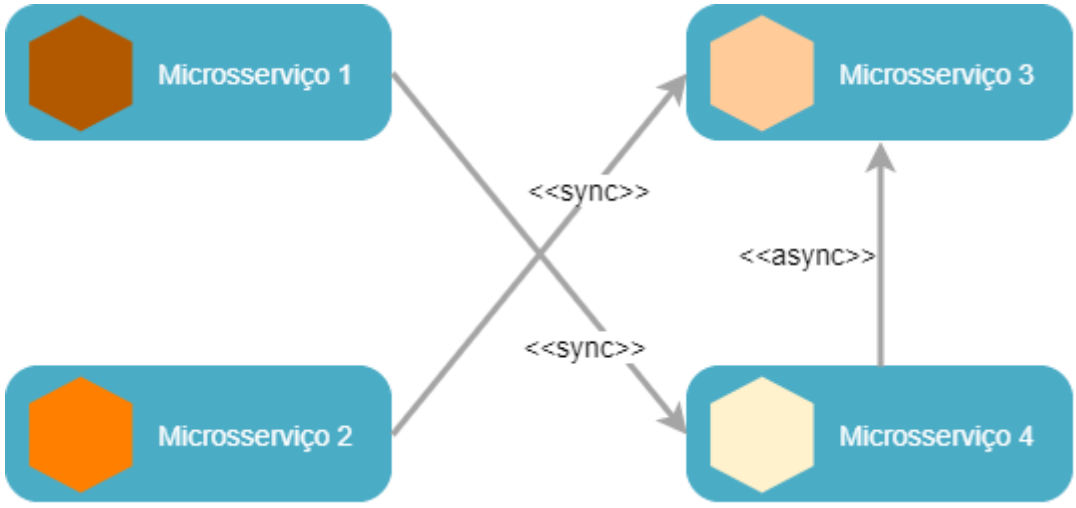

Figura 2 *- Interoperabilidade por meio de microsserviços*
Fonte: Elaboração Própria

2.1.1.2 Fraco Acoplamento

Uma das principais características de aplicações baseadas na arquitetura de microsserviços é o chamado baixo ou fraco acoplamento. Dentro do contexto deste trabalho, o fraco acoplamento possibilitou a construção de serviços menores e independentes, possuindo limites e responsabilidades bem definidos, seguindo o Princípio da Responsabilidade Única, ou *Single Responsibility Principle – SRP* (MARTIN, 2003).

O fraco acoplamento em microsserviços também demanda a descentralização da camada de armazenamento de dados, na qual cada serviço pode eleger seu

próprio mecanismo, sendo assim independente à tecnologia (banco de dados relacional ou não relacional, por exemplo) diferente da tradicional arquitetura monolítica. A figura 3 traça um paralelo entre o uso de banco de dados entre as arquiteturas monolítica e de microsserviços.

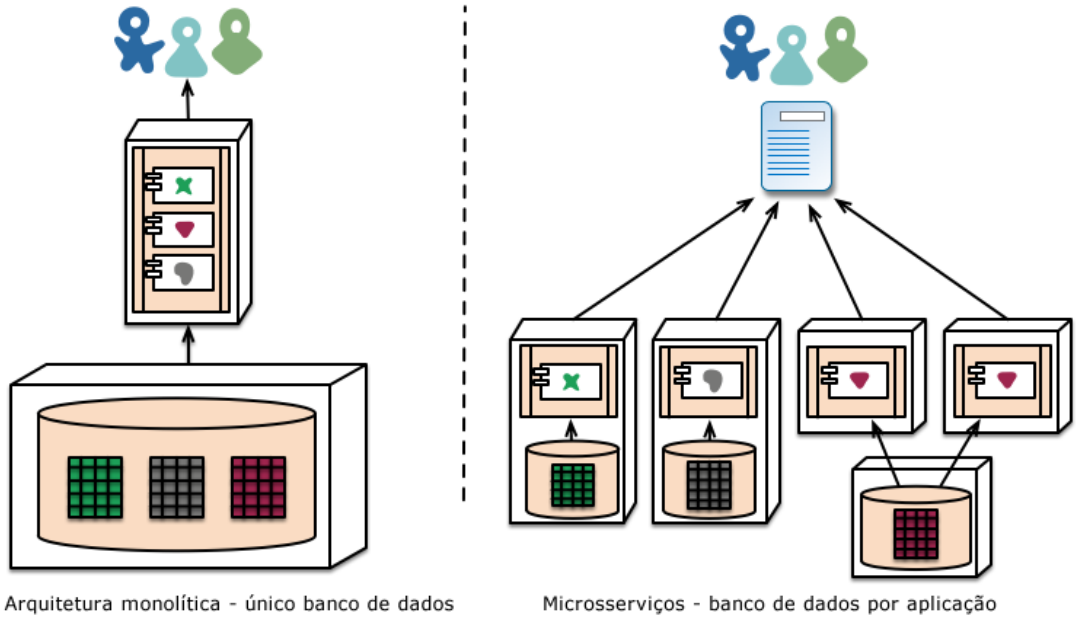

Figura 3 *- Comparativo de banco de dados em arquitetura monolítica e microsserviços*

Fonte: Adaptado de Fower, Lewis 2014

2.1.1.3 Tolerância a Falhas e Recuperabilidade

Devido a sua característica de organização por meio de diversos componentes, com responsabilidades claras e distintas, é natural verificar que microsserviços possuam tolerância a falhas nativas. Tal característica, entretanto, deve ser levada em conta durante todo o projeto de desenvolvimento: serviços podem falhar a qualquer momento e isso deve ser uma premissa (HAMORI, 2016) e devem ser projetados de modo a lidar com falhas corretamente; por exemplo, implementando padrões de arquitetura de resiliência como o de interrupção de circuito – *circuit breaker pattern* (MICROSOFT, 2019). A figura 4 ilustra como o padrão de interrupção de circuito possibilita o isolamento de falhas, sem impactar nos demais serviços.

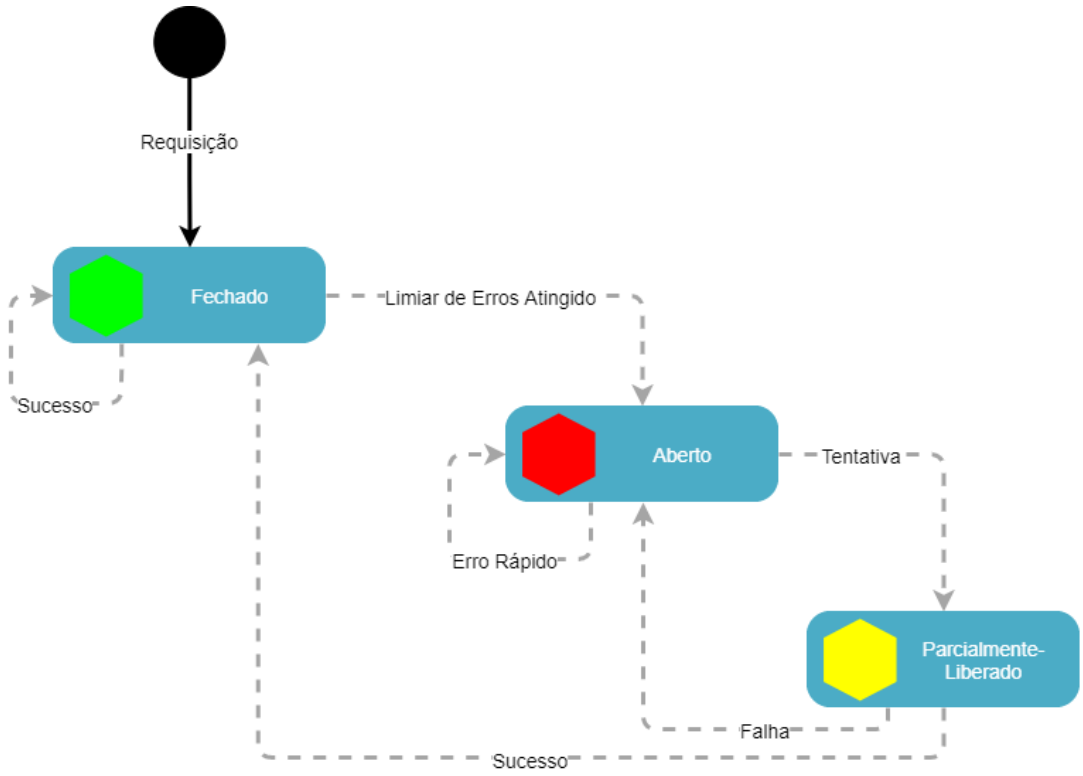

Figura 4 *- Padrão de Resiliência Interrupção de Circuito*

Fonte: Elaboração Própria

2.1.1.4 Escalabilidade

Microsserviços possibilitam que se escale horizontalmente serviços que demandem mais recursos, sem a necessidade de escalar horizontalmente o aplicativo como um todo.

Por meio de um orquestrador/gerenciador de contêineres, como o Kubernetes, é possível empacotar uma densidade maior de serviços, permitindo sempre uma utilização mais eficiente de recursos (MICROSOFT, 2019).

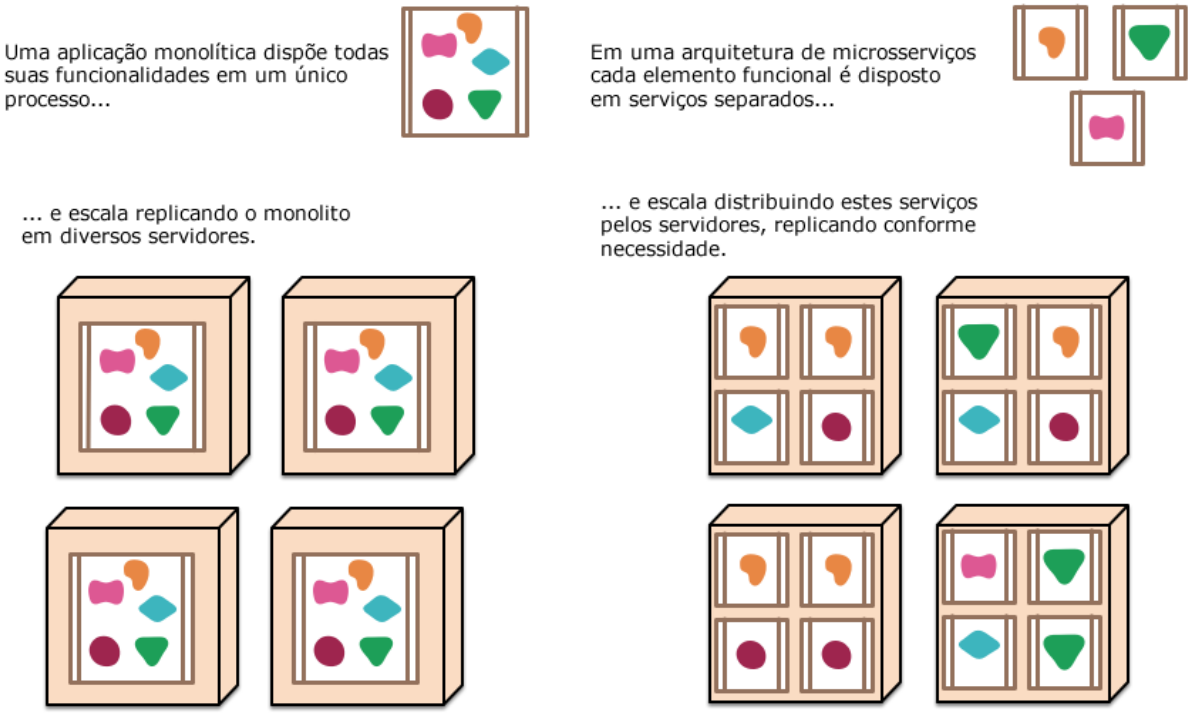

Figura 5 *- Comparativo de escalabilidade em arquitetura monolítica e microsserviços*

Fonte: Adaptado de Fower, Lewis 2014

2.1.2 Tecnologias

É importante frisar inicialmente que a arquitetura de microsserviços, por si só, não é uma tecnologia. São seus padrões (tecnologias) que lhe oferecem o suporte que a viabiliza. Em grande parte dos casos, implementações baseadas em microsserviços fazem uso de *APIs REST* e suas tecnologias de apoio, como os gerenciadores de contêineres, o Kubernetes e o *JavaScript Object Notation* – JSON, para representação das entidades de negócio.

Diante do exposto, serão abordadas a seguir as principais - e talvez fundamentais - tecnologias que microsserviços empregam.

2.1.2.1 Application Programming Interface *(API)*

API é um acrônimo em inglês que significa interface de programação de aplicações. Esta interface de programação é um conjunto de padrões de programação

que possibilitam a construção de aplicativos, a comunicação entre eles e sua utilização sem a necessidade de possuir conhecimento da maneira como foram implementados (PRESSMAN, 2016).

APIs também simplificam a forma como os desenvolvedores integram novos componentes de aplicações a uma arquitetura preexistente. Em razão disso, elas ajudam na colaboração entre as empresas e as equipes de TI. (RED HAT, 2020).

2.1.2.2 REST e RESTful

REST (*REpresentational State Transfer*) é o estilo arquitetural proposto por Roy Thomas Fielding, em 2000, em sua dissertação de doutorado.

REST indica um conjunto coordenado de restrições (*constraints*) arquiteturais que permite o baixo acoplamento entre os componentes e minimização da latência na comunicação, favorecendo também a sua escalabilidade (FIELDING, 2000):

- **Cliente-Servidor:** a restrição cliente-servidor permite separar as preocupações de interface com o usuário e armazenamento de dados, por exemplo.
- **Interface Uniforme:** simplifica e desacopla a arquitetura, permitindo que cada parte evolua independentemente, por meio de mensagens auto descritivas, por exemplo.
- **Sistema em Camadas:** camadas adicionais para balanceamento de carga ou segurança podem ser utilizadas para melhorar a escalabilidade e gestão de acessos.
- **Armazenável em Cache:** respostas podem ser armazenadas em cache, visto que um cache bem gerenciado elimina parcial ou completamente algumas interações cliente-servidor, melhorando ainda mais a escalabilidade e o desempenho.

- **Sem Estado (Stateless):** o servidor não deverá manter nenhum tipo de estado entre duas ou mais requisições. Cada solicitação de qualquer cliente deve conter todas as informações necessárias para ser atendida.

Por fim, uma API que implemente os princípios de restrições de Fielding é dita “RESTful”; ou seja, enquanto REST refere-se a um estilo arquitetural formalmente definido, RESTful refere-se à capacidade de um sistema de adotar estes princípios.

2.1.2.3 *JavaScript Object Notation* (JSON)

Apesar de Fielding (2000) não definir um formato, existe no JSON (*JavaScript Object Notation*) um consenso de mercado para representação, armazenamento e transmissão de informações em uma abordagem de microsserviços (JUNIOR, 2019).

A ideia utilizada pelo JSON para representação de informações é a de chave-valor, sendo uma sintaxe derivada do próprio *JavaScript*. Um par chave-valor deve ser representado pelo seu nome, entre aspas, seguido de dois pontos, seguido do seu valor. Os valores possuem três tipos básicos: numérico (inteiro ou real), booleano e caractere (*string*).

A figura 6 ilustra o JSON de uma venda parcelada em três vezes, documento este que representa uma das entidades de negócio do estudo de caso e será explorado em outros capítulos.

```
{
   "nsu":4564654,
   "data":"21/07/2020 17:00:00",
   "valor":450.00,
   "status":"ACEITO",
   "modalidadeVenda":"CREDITO",
   "modalidadePagamentoVenda":"CREDITO_A_VISTA",
   "parcelas":[
       {"numero": 1, "status": "PAGO", "valor": 150.00},
       {"numero": 2, "status": "PAGO", "valor": 150.00},
       {"numero": 3, "status": "PENDENTE", "valor": 150.00}
   ],
   "pontoVenda":{
     "codigo":1,
     "nomeFantasia":"EMPRESA LTDA",
     "cnpj":312131321
   }
}
```


Figura 6 *- Exemplo de um Documento JSON*

Fonte: Elaboração Própria

2.1.2.4 Swagger

Swagger é uma tecnologia de código aberto definida pela especificação *OpenAPI*, atualmente mantida pela *Open API Initiative*. É por meio dessa especificação que serviços podem ser descritos de forma independente da plataforma e linguagem de programação.

Swagger ainda tem como principal objetivo padronizar as APIs REST, descrevendo os recursos que uma determinada API deve possuir, bem como seus *endpoints*, estrutura de dados de recebimento e de retorno, códigos HTTP, métodos de autenticação, entre outros. A especificação do Swagger pode ser realizada automaticamente por meio de anotações (*annotations*) no código-fonte, aumentando assim a produtividade no desenvolvimento de APIs (OPENAPI INITIATIVE et al, 2017).

A figura 7 ilustra a representação visual para consumo de uma API de vendas, enquanto a figura 8 ilustra o trecho de um documento Swagger baseado em JSON para busca de uma venda.

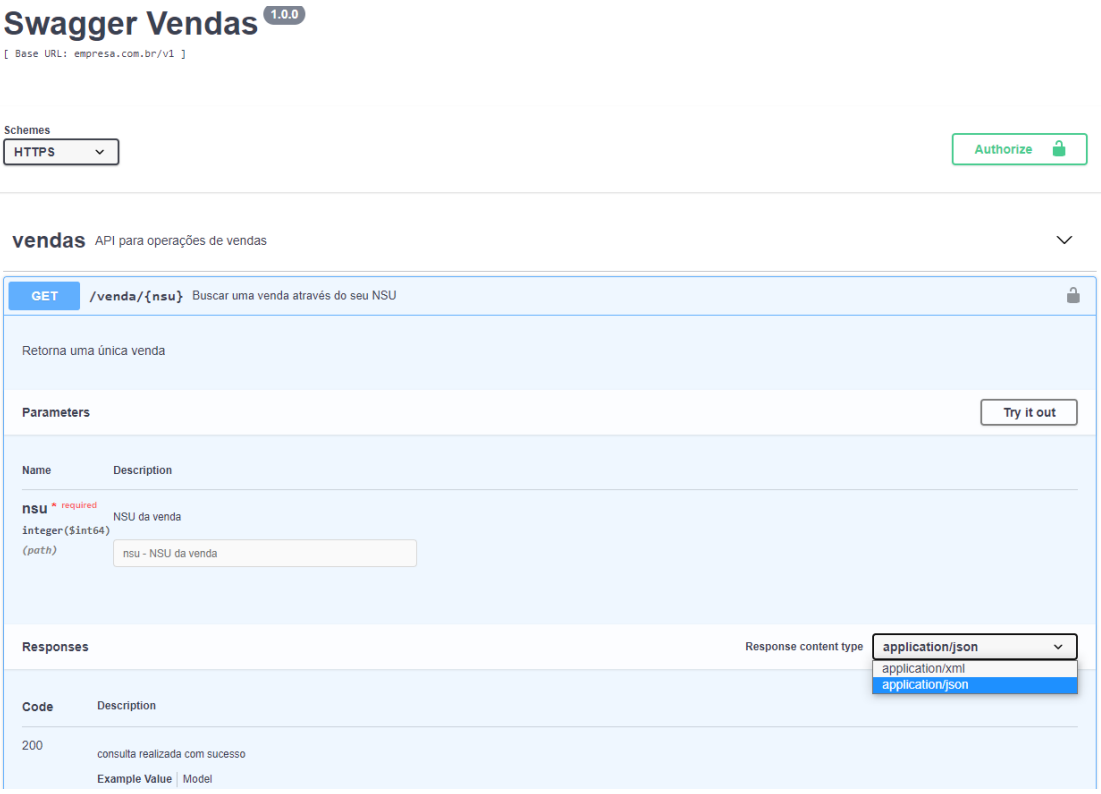

Figura 7 *- Representação Visual de um Documento Swagger*

Fonte: Elaboração Própria

```
{
  "swagger": "2.0",
  "info": {
    "description": "",
    "version": "1.0.0",
    "title": "Swagger Vendas"
  },
  "host": "empresa.com.br",
  "basePath": "/v1",
  "tags": [
    {
      "name": "vendas",
      "description": "API para operações de vendas"
    }
  ],
  "schemes": [
    "https"
  ],
  "paths": {
    "/venda/{nsu}": {
      "get": {
        "tags": [
          "vendas"
        ],
        "summary": "Buscar uma venda através do seu NSU",
        "description": "Retorna uma única venda",
        "operationId": "getVendaByNSU",
        "produces": [
          "application/xml",
          "application/json"
        ],
        "parameters": [
          {
            "name": "nsu",
            "in": "path",
            "description": "NSU da venda",
            "required": true,
            "type": "integer",
            "format": "int64"
          }
        ],
        "responses": {
          "200": {
            "description": "consulta realizada com sucesso",
            "schema": {
              "$ref": "#/definitions/Venda"
            }
          },
```

Figura 8 *- Trecho de um documento Swagger*

Fonte: Elaboração Própria

2.1.2.5 Orquestrador de Contêiner

Em microsserviços, aplicações são compostas por componentes compactados individualmente em contêineres, que consistem em “um conjunto de um ou mais processos organizados isoladamente do sistema. Todos os arquivos necessários para executá-los são fornecidos por uma imagem distinta.” (RED HAT, 2020).

No processo moderno de desenvolvimento é natural que esses contêineres cheguem à ordem de dezenas ou mesmo centenas e precisem trabalhar em conjunto, para que uma determinada aplicação como um todo funcione adequadamente. Sendo assim, a orquestração de contêineres refere-se ao processo de organizar e gerenciar os diversos componentes de uma arquitetura de microsserviços (HEWLETT PACKARD, 2020).

2.2 O Cenário Atual

A arquitetura de microsserviços não é uma abordagem completamente nova, mas sim uma combinação de diversos métodos que foram bem-sucedidos na engenharia de software, como o desenvolvimento ágil, conceito de desenho de API primeiro (*API-first*), entre outros (AMAZON WEB SERVICES, 2019). Conforme vem sendo levantado no presente trabalho, a literatura especializada para microsserviços é bastante atualizada e estruturada, contemplando diversos pontos relevantes que auxiliam na tomada de decisão das organizações.

Além de definições de características e tecnologias inerentes à arquitetura de microsserviços, questões relacionadas à segmentação em níveis de maturidade bem definidos e estruturados, e uma boa separação dos tipos de serviços que encaixem em determinados cenários são fatores fundamentais para o uso efetivo do modelo. A seguir, será apresentado o modelo de maturidade de APIs proposto por Leonard Richardson.

2.2.1 Modelo de Maturidade de APIs de Richardson

Conforme visto no subcapítulo 2.1.2.2, Fielding propõe uma série de restrições para que uma API seja considerada RESTful. Tais restrições acabam sendo por vezes

complexas para a realidade, e questões como “Isso não é possível!” ou até mesmo “Esse modelo é inalcançável!” são levantadas com frequência (FUSCELLA, 2017).

Baseando-se nesse contexto, *Richardson* (2010), propõe um novo modelo de maturidade de APIs, também denominado como "RMM" (*Richardson Maturity Model* ou MMR - Modelo de Maturidade de Richardson). Por meio dele pode-se estabelecer em qual nível de maturidade determinada organização insere-se em relação ao emprego de APIs em microsserviços.

Como pode-se observar na figura 9 o modelo é, de fato, mais simples, apresentando quatro níveis para atingir a “Glória do REST” (Richardson, 2010).

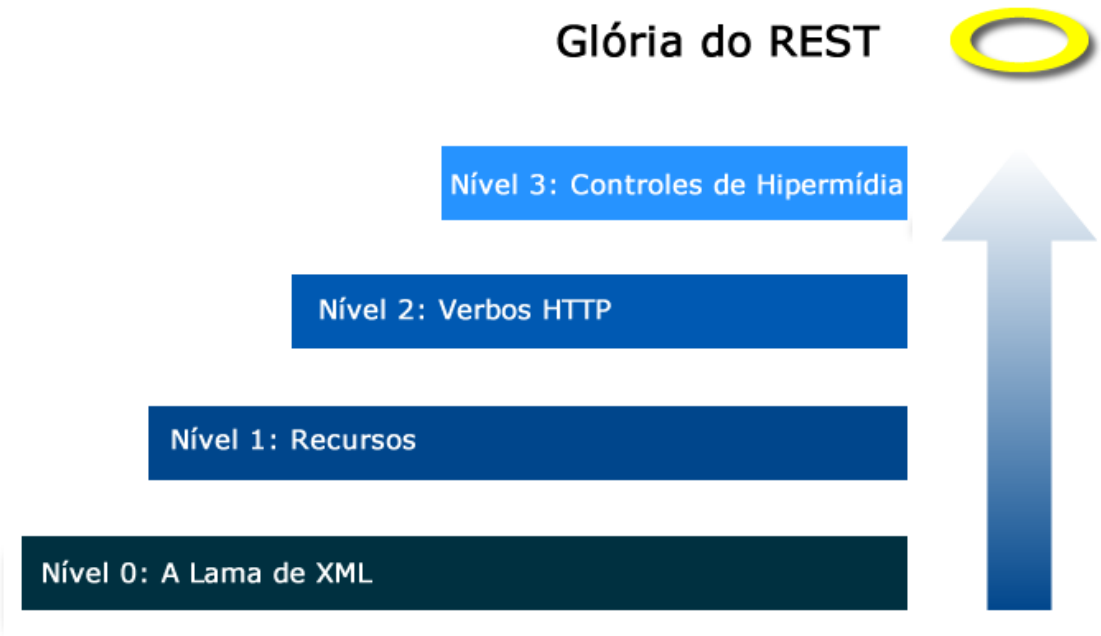

Figura 9 *- Modelo de Maturidade de Richardson*
Fonte: Adaptado de Richardson, 2010

Para exemplificar a diferença entre os níveis de maturidade será ilustrado o fluxo da consulta de uma venda e posterior cancelamento de uma parcela em aberto.

2.2.1.1 Nível 0 - A Lama de XML

O ponto de partida do modelo considera o uso do HTTP apenas como uma camada de transporte para interações remotas, neste nível a troca de mensagens

pode ser baseada em XML (o *plain old XML* ou “POX”, como o autor nomeia) ou até mesmo JSON. Arquiteturas baseadas em SOAP ou XML-RPC encontram-se neste nível (RICHARDSON, 2010).

A figura 10 ilustra um exemplo de interação de chamadas em uma API na maturidade 0.

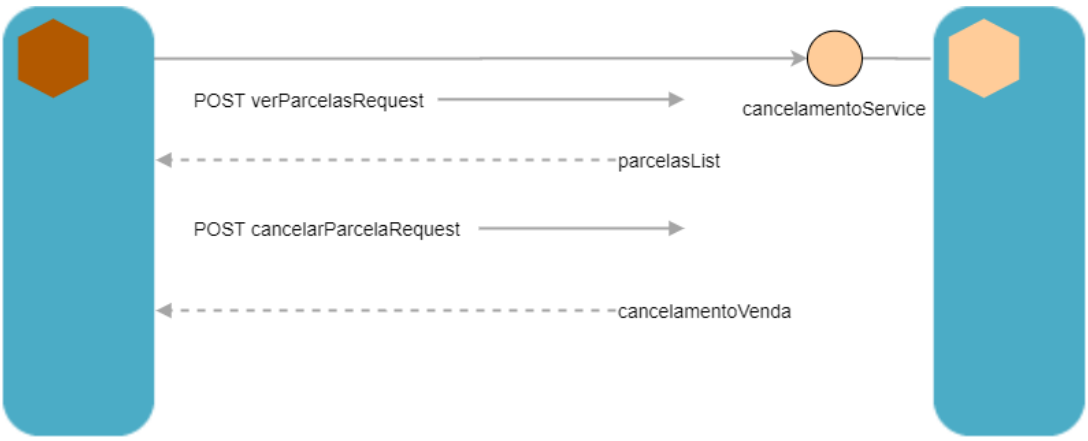

Figura 10 *- Exemplo de interação no nível 0 de Richardson*
Fonte: Adaptado de Richardson, 2010

Em um cenário de nível 0 verifica-se que o serviço se encontra exposto por meio de um único *endpoint* (cancelamentoService) no qual as operações para consulta e cancelamento de uma venda ou parcela variam apenas no corpo da mensagem enviada.

Em termos técnicos, todas as requisições ao serviço fazem uso do verbo POST e as respostas sempre retornam o código 200 – de sucesso – do HTTP, independente do cenário relacionado ao contexto de negócio (uma parcela que resulte na criação de um cancelamento, por exemplo).

POST /cancelamentoService HTTP/1.1
[demais cabeçalhos HTTP]
<cancelarParcelaRequest venda = 1234 parcela = 3/>

Assumindo um cenário em que a terceira parcela da venda 1234 tenha sido realizada com êxito, o serviço retorna o mesmo conteúdo por meio do código HTTP 200.

HTTP/1.1 200 OK
[demais cabeçalhos HTTP]
<cancelamento>
<venda = 1234 parcela = 3/>
</cancelamento>

2.2.1.2 Nível 1 - Recursos

É no nível 1 que apresenta-se efetivamente o primeiro passo para alcançar a "Glória do REST", aqui se tem a introdução de recursos; dessa forma as chamadas ao serviço deixam de ser realizadas por meio de um único *endpoint*, e recursos passam a ser tratados individualmente, tendo mais semântica para o próprio contexto das aplicações consumidoras, conforme visualiza-se na figura 11.

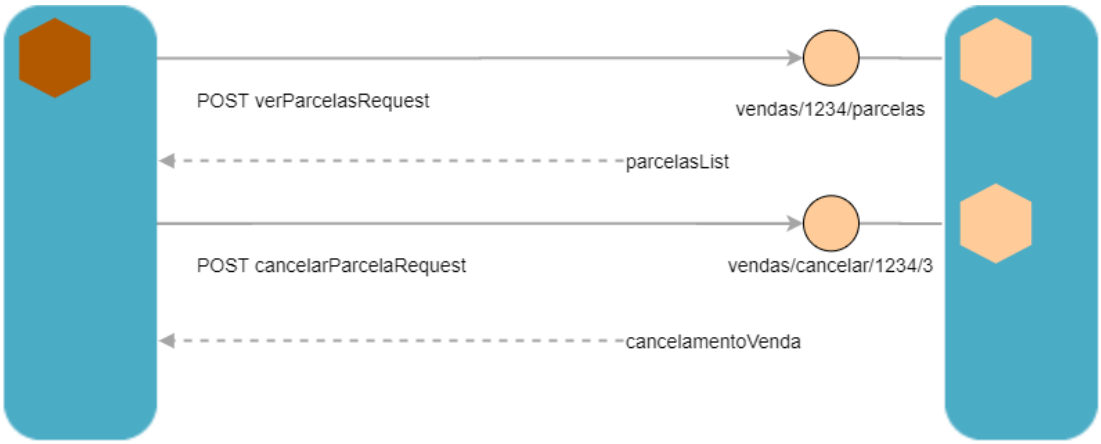

Figura 11 *- Interação no nível 1 de Richardson adiciona recursos*
Fonte: Adaptado de Richardson, 2010

Em termos técnicos há pouca mudança neste nível, conforme já exposto, as mudanças são essencialmente na disposição das interações baseadas em recursos,

sendo assim as mensagens trafegadas permanecem utilizando o verbo POST e recebendo o código 200 nos retornos.

POST /vendas/cancelar/1234/3 HTTP/1.1
[demais cabeçalhos HTTP]
<cancelarParcelaRequest venda = 1234 parcela = 3/>

HTTP/1.1 200 OK
[demais cabeçalhos HTTP]
<cancelamento>
<venda = 1234 parcela = 3/>
</cancelamento>

2.2.1.3 Nível 2 - Verbos HTTP

A partir deste nível, os verbos do HTTP passam a ter uma utilização mais aderente ao seu propósito de criação.

GET /vendas/1234/parcelas?status=PENDENTE HTTP/1.1
[demais cabeçalhos HTTP]

Operações sem alteração de estado passam a utilizar o verbo “GET”, agregando aqui vantagem de performance, por meio do uso de cache de dados. A figura 12 ilustra a sequência de chamadas para uma API de nível 2 de maturidade.

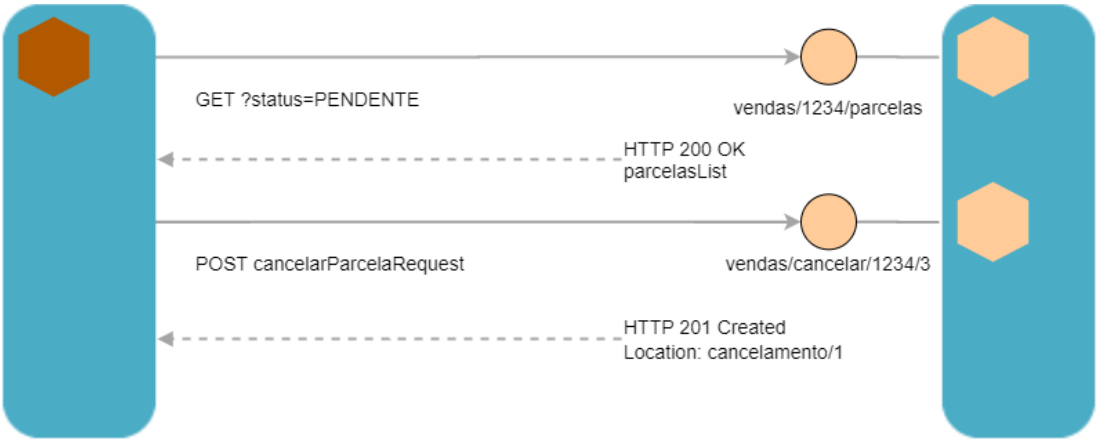

Figura 12 *- Interação no nível 2 de Richardson adiciona verbos HTTP*
Fonte: Adaptado de Richardson, 2010

É também no nível 2 que os códigos de retorno do HTTP passam a ter uma diferenciação. Um eventual erro no serviço pode retornar o código HTTP 500 e um sucesso na criação de uma operação de negócio pode retornar o código HTTP 201, sendo que este também inclui um atributo com a localização do objeto criado (*location*), que pode ser utilizada para obter o estado atual desse recurso futuramente. A resposta também inclui uma representação do recurso criado, poupando o cliente de uma requisição extra (RICHARDSON, 2010) e a partir deste nível será utilizado o JSON para esta representação.

```
POST /vendas/cancelar/1234/3 HTTP/1.1
[demais cabeçalhos HTTP]
{
    venda: 1234,
    parcela: 3
}

HTTP/1.1 201 Created
[demais cabeçalhos HTTP]
{
    cancelamentoId: 1,
    tipoCancelamento: “TOTAL”
    valorCancelamento: 150.00
    venda: 1234,
    parcela: 3
}
```

2.2.1.4 Nível 3 - Controles de Hipermídia

Por fim, no nível 3, há a Hipermídia Como Mecanismo de Estado da Aplicação (o “HATEOAS” - *Hypermedia as the Engine of Application State*), que Fielding também descreve em seu trabalho como uma das restrições para considerar uma API RESTful. Apresenta-se na figura 13 a sequência de chamadas para uma API de nível 3 de maturidade.

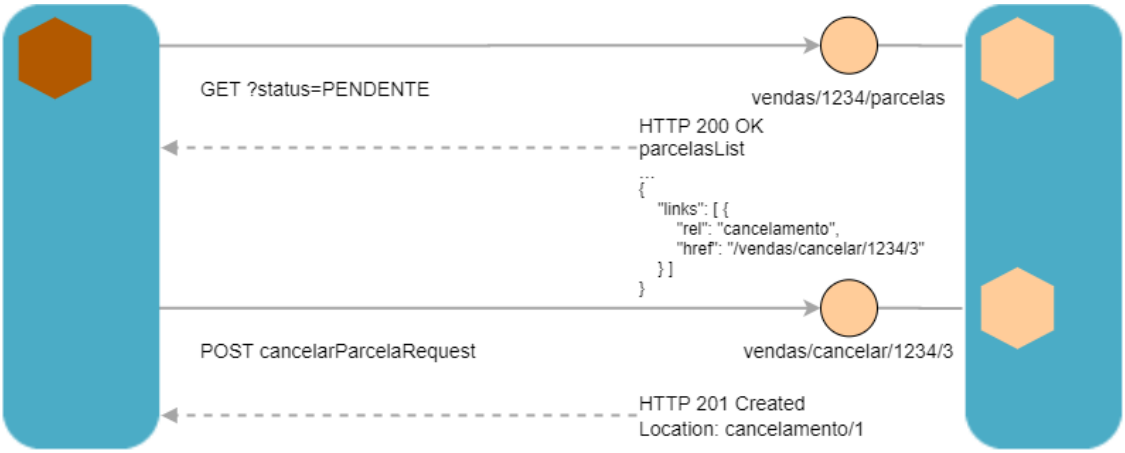

Figura 13 *- Interação no nível 3 de Richardson adiciona controles de hipermídia*

Fonte: Adaptado de Richardson, 2010

Aqui mantém-se a operação GET do nível 2.

GET /vendas/1234/parcelas?status=PENDENTE HTTP/1.1
[demais cabeçalhos HTTP]

Entretanto há um novo e importante elemento em sua resposta:

HTTP/1.1 200 OK
[demais cabeçalhos HTTP]
{
"nsu": 4564654,
"data": "21/07/2020 17:00:00",
"valor": 450.00,
"status": "ACEITO",
"modalidadeVenda": "CREDITO",
"modalidadePagamentoVenda": "CREDITO_A_VISTA",
"parcelas": [{
"numero": 3,
"status": "PENDENTE",
"valor": 150.00,
"links": [{
"rel": "self",
"href": " /vendas/1234"
}, {
"rel": "cancel",
"href": " /vendas/cancelar/1234/3"

```
        }]
      }],
      "pontoVenda": {
        "codigo": 1,
        "nomeFantasia": "EMPRESA LTDA",
        "cnpj": 312131321
      }
}
```

Nota-se agora que uma venda possui um correspondente de hipermídia (atributo “*links*” do JSON) que, além de apontar para si, também aponta para uma possibilidade de estado (cancelamento de uma parcela pendente). É nesse contexto que se apresenta o ponto mais importante do HATEOAS: a forma como um recurso pode ser manipulado é autoexplicativa, evitando assim que o cliente precise “adivinhar” quais operações estão habilitadas (RICHARDSON, 2010).

2.2.1.5 O Significado dos Níveis

Richardson conclui o artigo afirmando que o seu modelo fornece boas práticas para o entendimento de como chegar em uma abordagem RESTful, embora não seja uma definição dos níveis do próprio REST, visto que Fielding (2000) já havia definido em seu trabalho que o nível 3 seria uma precondição.

Ian Robison (2010) ainda ressalta que o modelo é atraente para uso prático devido ao seu relacionamento direto com técnicas comuns de design, técnicas essas que também servirão como uma das bases metodológicas deste trabalho e serão mais bem exploradas no capítulo 3.2.

2.3 Microsserviços: Um Breve Comparativo

Neste capítulo será realizado um breve comparativo entre as abordagens da Arquitetura Orientada a Serviços (SOA) e a Arquitetura de Microsserviços, termos estes que ainda geram certo conflito e desentendimentos. Espera-se ainda poder desmistificar a ideia de que exista uma possível rivalidade direta entre os conceitos.

2.3.1 Microsserviços e a Arquitetura Orientada a Serviços (SOA)

Para que seja possível traçar o paralelo entre SOA e microsserviços é importante definir brevemente o que é SOA: a arquitetura orientada a serviços, por si só, consiste em um estilo arquitetural que prega, fundamentalmente, que as funcionalidades das aplicações devem ser disponibilizadas na forma de serviços. LUBLINSKY (2007) ainda ressalta que, em grande parte dos cenários, “estes serviços estão conectados em uma solução conhecida como barramento de serviços corporativos (ESB - *enterprise service bus*), disponibilizando interfaces ou contratos acessíveis através de *webservices* ou uma outra forma de comunicação entre aplicações.”

2.3.1.1 Convergências

SOA surgiu com objetivo de realizar integrações entre aplicações e trazer uma visão alinhada ao negócio, para que as organizações pudessem responder mais rápido a mudanças, suportando diferentes plataformas e protocolos (ERL, 2009). O autor traça em seu trabalho o chamado “princípio do design de serviço”, que pode ser representado nos seguintes tópicos:

- Serviços são reutilizáveis;
- Serviços compartilham um contrato formal;
- Serviços possuem um baixo acoplamento;
- Serviços abstraem a lógica;
- Serviços são capazes de se compor;
- Serviços são autônomos;

- Serviços evitam alocação de recursos por longos períodos.

JAMSHIDI, Pooyan et al (2018) ainda traçam uma linha do tempo para microsserviços na qual SOA é um dos elementos de fundamento, conforme pode-se ver na figura 14. Sendo SOA um dos alicerces, verifica-se que microsserviços ainda exigem uma definição e padronização mais aprofundada e compatível com sua natureza dinâmica (JUSTINO, 2018).

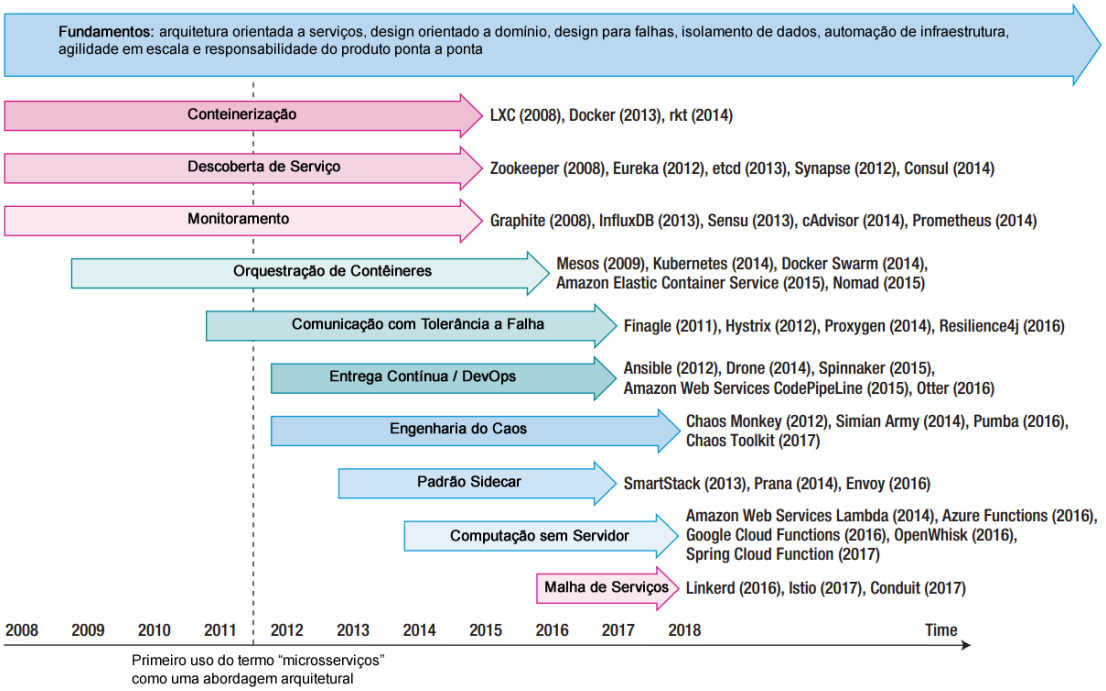

Figura 14 *- Linha do tempo associada a microsserviços*
Fonte: Adaptado de JAMSHIDI, Pooyan et al (2018)

2.3.1.2 Diferenças

NEWMAN (2015) enfatiza que, apesar da falta de consenso sobre como fazer SOA bem, microsserviços surgiram por meio de práticas constatadas no mundo real, tendo assim uma melhor compreensão, inclusive impulsionando o surgimento de outras tecnologias, conforme pode-se observar na figura 14.

As diferenças entre microsserviços e SOA residem basicamente em aspectos tecnológicos e de processos organizacionais e de desenvolvimento, conforme pode-se observar na tabela 1.

Tabela 1 *- Comparativo microsserviços e SOA*

	Microsserviços	**SOA**
Comunicação	Faz uso de protocolos leves e abertos para comunicação entre os componentes, geralmente baseado em APIs REST sob HTTP.	Utiliza o barramento de serviços corporativo (ESB), permitindo múltiplos protocolos e transformação de dados.
Tamanho dos Componentes	Sempre pequeno, com partes das lógicas de negócio distribuídas entre os componentes.	Geralmente grandes, tendo toda lógica de um negócio em um único componente.
Armazenamento de Dados	Cada microsserviço possui mecanismo de armazenamento próprio.	Geralmente serviços compartilham a camada de armazenamento de dados.
Governança	Governança flexível, com foco na colaboração de equipe e liberdade de escolha.	Foco em governança centralizada e padronizada.
Estrutura Organizacional	Times pequenos e multidisciplinares.	Qualquer uma, geralmente times médios com foco em desenvolvimento de software.

Fonte: Adaptado de ARSOV (2017)

2.3.1.3 Conclusão

Conforme exposto nos capítulos anteriores, microsserviços possuem de fato um alicerce na Arquitetura Orientada a Serviços. Boas práticas relacionadas ao design de serviços e o pensamento alinhado aos processos de negócios das organizações são tópicos baseados nos princípios SOA que microsserviços de fato aplicam.

Cabe observar que a literatura de microsserviços vai além: ela oferece também uma abordagem tática, que faz uso de práticas modernas da engenharia de software (apesar de muitos conceitos relacionados não terem surgido de fato por meio de microsserviços) fazendo assim o emprego deste modelo ser uma convergência dos princípios de SOA, sustentados com tecnologias abordadas em microsserviços.

3 Metodologia

3.1 Metodologia do Estudo de Caso e Projeto

O trabalho também se baseia em um estudo de caso particular de um projeto de sistema de informação, que teve como desafio evoluir uma arquitetura monolítica para uma arquitetura baseada em microsserviços, tendo como um dos seus objetivos alcançar o nível 3 do Modelo de Maturidade de Richardson.

Em seu artigo, Richardson (2010) afirma que Ian Robinson vê atração em seu modelo devido a sua relação direta com técnicas comuns de design, criando assim um modelo de referência sobre qual tipo de serviço oferecer e proporcionar às pessoas que queiram interagir com ele, alinhando expectativas. As etapas adotadas neste estudo de caso estão demonstradas na figura 15.

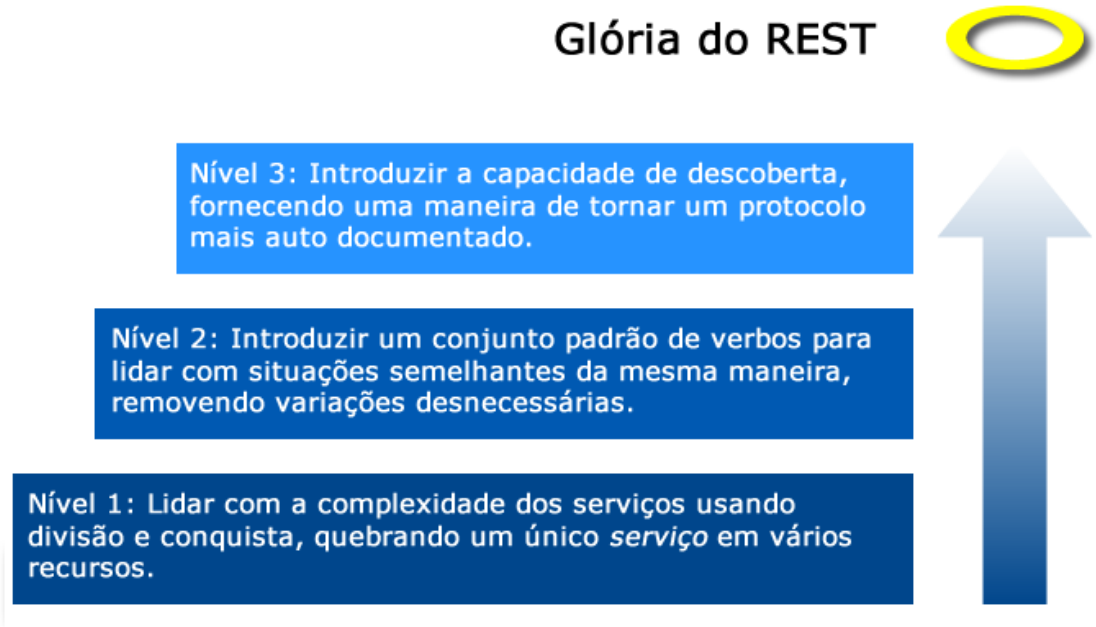

Figura 15 - *Etapas para a evolução/avaliação da maturidade de Richardson*
Fonte: Adaptado de Richardson (2010)

Por meio destas etapas foi possível iniciar a decomposição da atual arquitetura monolítica para uma solução baseada em microsserviços. Os detalhes de como esta migração foi realizada serão apresentados no capítulo 4.1.3.

O projeto desenvolvido também fez uso da metodologia ágil *Scrum*, que em sua fase de “pré-jogo” (*pre-game phase*) prioriza e estima os requisitos do produto, possibilitando o cálculo do ROI (sigla do inglês *Return on Investiment* ou Retorno Sobre o Investimento), além de englobar outras atividades essenciais, como a definição e formação da equipe multidisciplinar que irá desenvolver o produto. É também nessa fase que a proposta de arquitetura do sistema é elaborada, bem como a identificação de ferramentas a serem utilizadas, possíveis riscos e necessidades de treinamento (SABBAGH, 2014), conforme observa-se na figura 16.

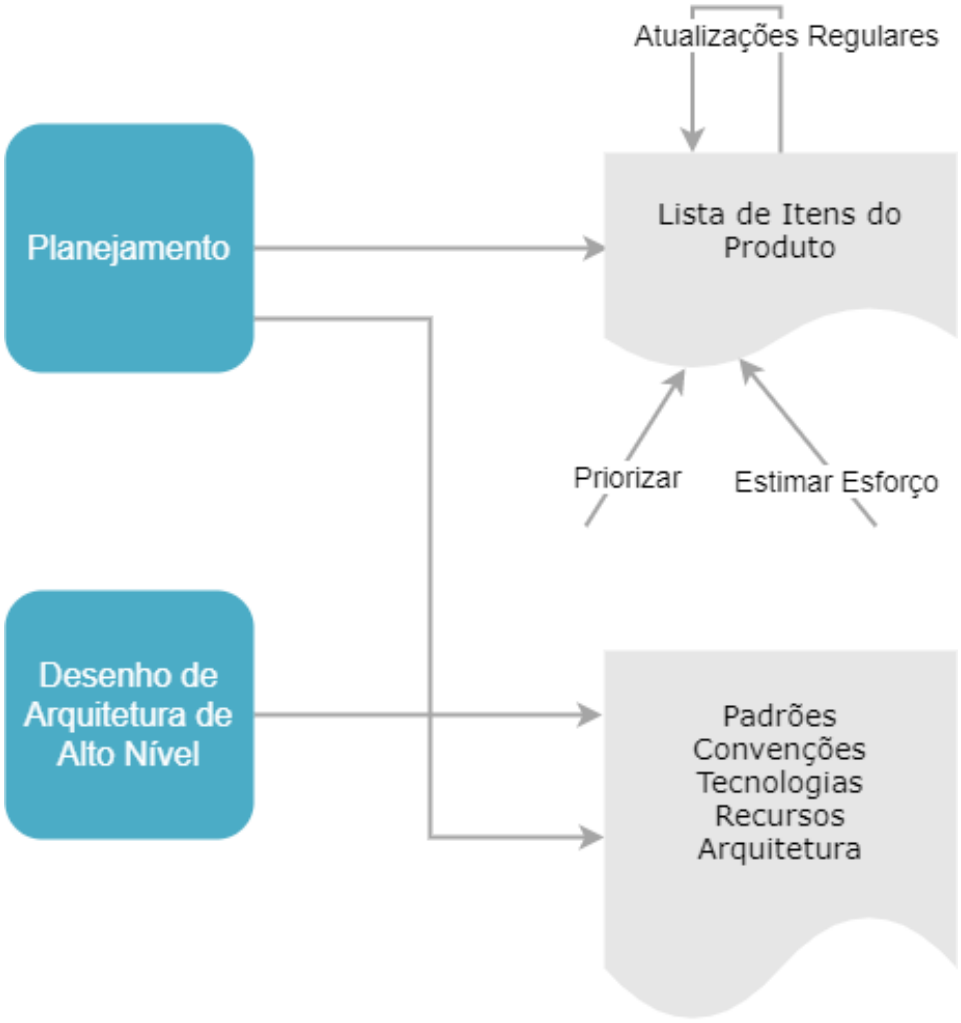

Figura 16 - *Fase de pré-jogo da metodologia ágil Scrum*

Fonte: Adaptado de Schwaber (2002)

Foi por meio das ações existentes na fase de pré-jogo do *Scrum* que foi possível desenhar a arquitetura de microsserviços proposta, além de criar e definir novos padrões de desenvolvimento, bem como estimar custos e esforços.

4 Resultados e Discussão

Os resultados apresentados a seguir são baseados em um projeto para a adoção de solução, baseada em microsserviços, em um processo de negócio de cancelamento de vendas, em uma empresa de meios de pagamento, que será nomeada a partir deste ponto de “Pagamentos Alfa”. O presente trabalho também teve a liberdade de inserir outros cenários de caos no produto, para uma melhor ilustração da aplicabilidade da arquitetura de microsserviços no projeto.

4.1 Estudo de Caso

4.1.1 A Empresa Pesquisada

A Pagamentos Alfa foi fundada há mais de 20 anos e emprega diretamente cerca de 2 mil funcionários, está presente em mais de 1,3 milhões de pontos de venda no país, transacionando mais de R$ 34 bilhões por mês.

O trabalho na fase de pré-jogo no projeto da Pagamentos Alfa consistiu na elaboração da nova proposta de arquitetura da aplicação, e também na formação de um time multidisciplinar com foco no produto – um dos fundamentos de microsserviços – para a criação de um plano de negócio contemplando os novos aspectos técnicos e de usabilidade do projeto versus o custo de sustentação da aplicação atual; bem como custos operacionais que poderiam ser resolvidos dentro da solução proposta (licenciamento do *Windows Server* e alto número de ligações na central de atendimento, por exemplo). Apresentadas essas variáveis, foi possível calcular o ROI estimado do projeto, conquistando desta forma o patrocínio da organização.

4.1.2 Visão da Arquitetura Atual

Atualmente o serviço para cancelamento de vendas da Pagamentos Alfa é realizado por três grupos distintos de usuários:

- **Lojistas:** cliente pessoa física ou jurídica da Pagamentos Alfa. Realizam suas vendas por meio das maquininhas de cartão e os cancelamentos destas pelo Portal do Cliente, site acessível via internet por meio de usuário e senha.

- **Empresas de Comércio Eletrônico:** empresas que realizam todas as suas vendas por meio da internet. Os cancelamentos são feitos pelo Portal do Cliente ou a partir de um arquivo de texto no formato CSV (da sigla *Comma-separated values*, ou seja, valores separados por vírgula) enviado por meio do Protocolo de Transferência de Arquivos (o *File Transfer Protocol* - FTP), também por meio de usuário e senha. Atualmente o processo de integração por FTP mostra-se defasado em relação ao mercado, ocasionando até perdas de contratos.

- **Operadores da Central de Atendimento:** funcionários diretos da Pagamentos Alfa, realizam o atendimento telefônico dos lojistas e das empresas de comércio eletrônico. Podem realizar o cancelamento de uma venda por meio do Portal do Funcionário, aplicação acessível via intranet por meio da matrícula e senha do funcionário; nesta aplicação o cancelamento também pode ser realizado por meio do arquivo CSV gerado pelas empresas de comércio eletrônico.

A figura 17 ilustra em detalhes o modelo atual dessa arquitetura, na qual também pode-se destacar os seguintes componentes:

- **Portal do Cliente:** interface acessada pelos lojistas e empresas de comércio eletrônico. Permite consultar as vendas realizadas e fazer seu respectivo cancelamento, entre outras funcionalidades de gestão de caixa. Em termos de experiência do usuário, uma venda é consultada em uma área do portal e o cancelamento é realizado em outra, gerando dúvidas e reclamações sobre a navegação.

- **Servidor FTP:** servidor responsável pela recepção dos arquivos CSV com lotes de cancelamentos enviados pelas empresas de comércio eletrônico e pelos operadores da central de atendimento. Está exposto a vulnerabilidades de segurança conhecidas neste protocolo, como a falta de criptografia dos dados, por exemplo.

- **Portal do Funcionário:** aplicação acessada dentro do ambiente de intranet da organização. Além do cancelamento comum ao Portal do

Cliente, permite também o envio de arquivos CSV para cancelamentos em lote.

- ***CancelamentoService*:** aplicação central do estudo de caso, sendo a responsável por efetivar um cancelamento recebido e notificá-lo ao cliente. Também acumula as responsabilidades de realizar as buscas de vendas, gravação de registros para auditoria e validação de lista restritiva para um cancelamento (para fins de prevenção a fraudes, determinados endereços IP e estabelecimentos não podem cancelar uma venda). Devido a sua criticidade, roda em um servidor *Windows* dedicado; entretanto, devido à característica monolítica, ainda é suscetível a paradas gerais por eventuais manutenções ou evoluções em código-fonte ou mesmo excesso de tráfego.

- ***CancelamentoBatch*:** aplicação responsável por buscar arquivos no servidor FTP, para posterior validação de seu conteúdo e cancelamento por lotes por meio de integração com a aplicação *CancelamentoService*. Em alguns períodos acaba indisponibilizando a aplicação principal, devido ao alto número de transações das empresas de comércio eletrônico. Roda em um servidor *Windows* dedicado.

- **Banco de Dados Corporativo:** banco de dados relacional de uso corporativo. Responsável pelo armazenamento e consulta de dados referentes às vendas realizadas, aos cancelamentos, registros de auditoria e ao cadastro de lista restritiva. Devido à concorrência de diferentes finalidades também sofre com eventuais paradas.

- ***MailService*:** serviço de notificação via e-mail para as empresas de comércio eletrônico que realizam cancelamentos em lote. Devido à característica de processamento em lotes, este serviço fica responsável por informar ao cliente o sucesso ou erro de um arquivo enviado.

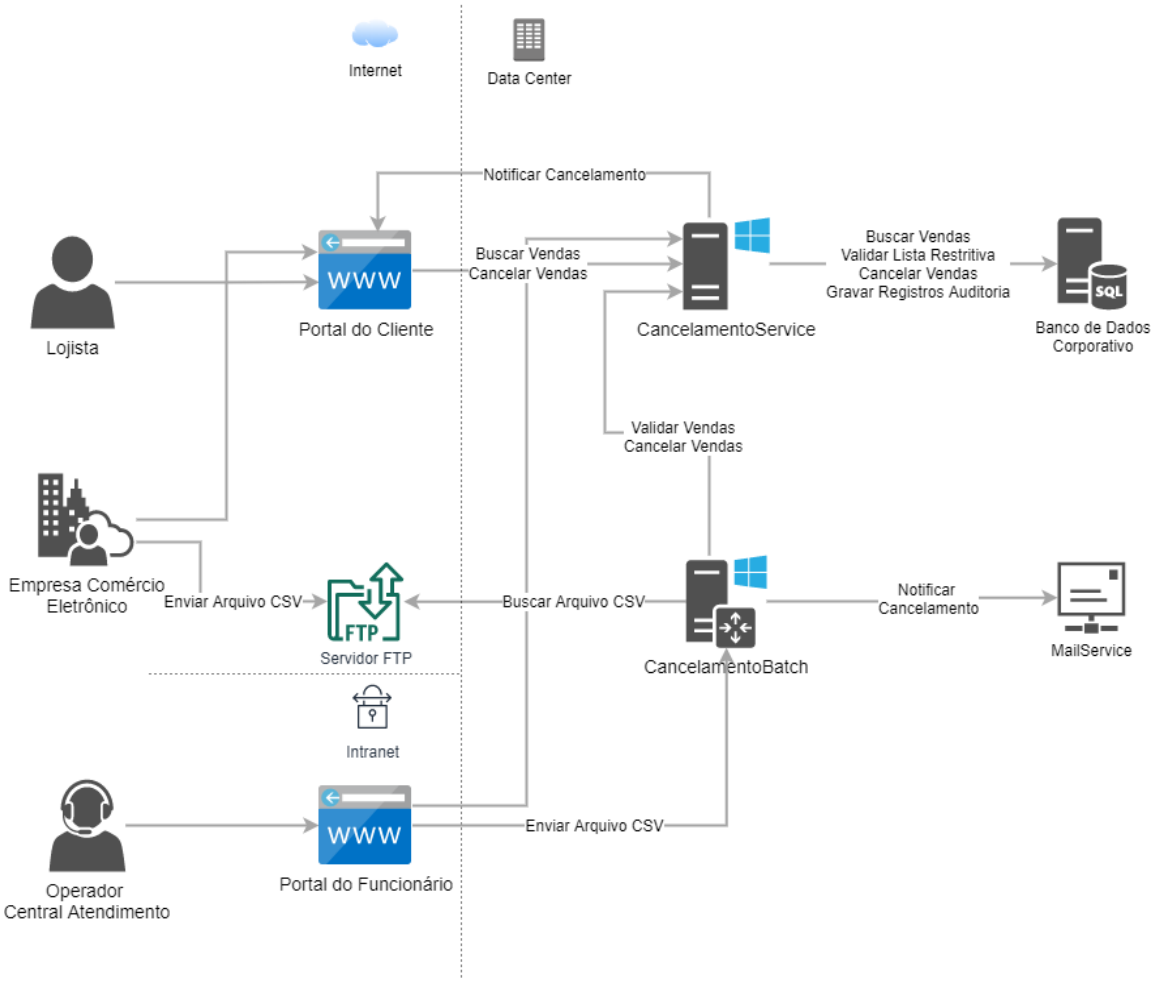

Figura 17 *- Arquitetura da Aplicação - Cenário Atual*

Fonte: Elaboração própria

Diante do cenário exposto pode-se elencar em tópicos os principais problemas enfrentados pela Pagamentos Alfa, diante da sua arquitetura atual de aplicação:

- Indisponibilidade geral de processos conceitualmente distintos, como consulta e cancelamento de vendas, devido à característica monolítica do serviço;
- Custo e risco elevado nas manutenções e evoluções sistêmicas devido à característica monolítica do serviço;
- Vulnerabilidades de segurança devido ao uso de um servidor FTP para transferência de arquivos;

- Vulnerabilidades de segurança na camada dos serviços expostos pela aplicação. Apesar da autenticação existente nos portais, eventuais chamadas diretas ao serviço podem ser exploradas;
- Perda de oportunidades de negócio, por não possuir uma plataforma de fácil integração para novas parcerias com empresas de comércio eletrônico.

4.1.3 Visão da Arquitetura Proposta

Expostos os principais problemas vivenciados pela organização diante da arquitetura atual e com o suporte do estudo do presente trabalho, a arquitetura de microsserviços mostrou-se como uma decisão consistente para a construção do novo desenho de arquitetura. A tabela 2 realiza um paralelo entre as características de microsserviços estudadas e sua relação direta de apoio e mesmo solução aos problemas elencados na seção anterior.

Tabela 2 *- Problemas da Arquitetura Atual x Características de Microsserviços*

	Características de microsserviços				
Problemas	**Interoperabilidade**	**Fraco Acoplamento**	**Tolerância a Falhas e Recuperabilidade**	**Escalabilidade**	***Gateway* de API**
Indisponibilidade geral de processos conceitualmente distintos		X	X	X	
Custo e risco elevado nas manutenções e evoluções		X			
Vulnerabilidades de segurança (FTP)	X				X
Vulnerabilidades de segurança (Serviços)					X
Perda de oportunidades de negócio	X				

Fonte: Elaboração própria

Considerando os problemas levantados e tendo como um dos objetivos do projeto atingir o nível 3 do modelo de maturidade de Richardson, a primeira ação foi lidar com a complexidade do serviço *CancelamentoService* por meio da divisão e

conquista, quebrando seu único serviço em vários (RICHARDSON, 2010) possibilitando assim um fraco acoplamento e facilidade para a escalabilidade, as manutenções e evoluções independentes. A figura 18 ilustra o cenário proposto para a arquitetura da aplicação.

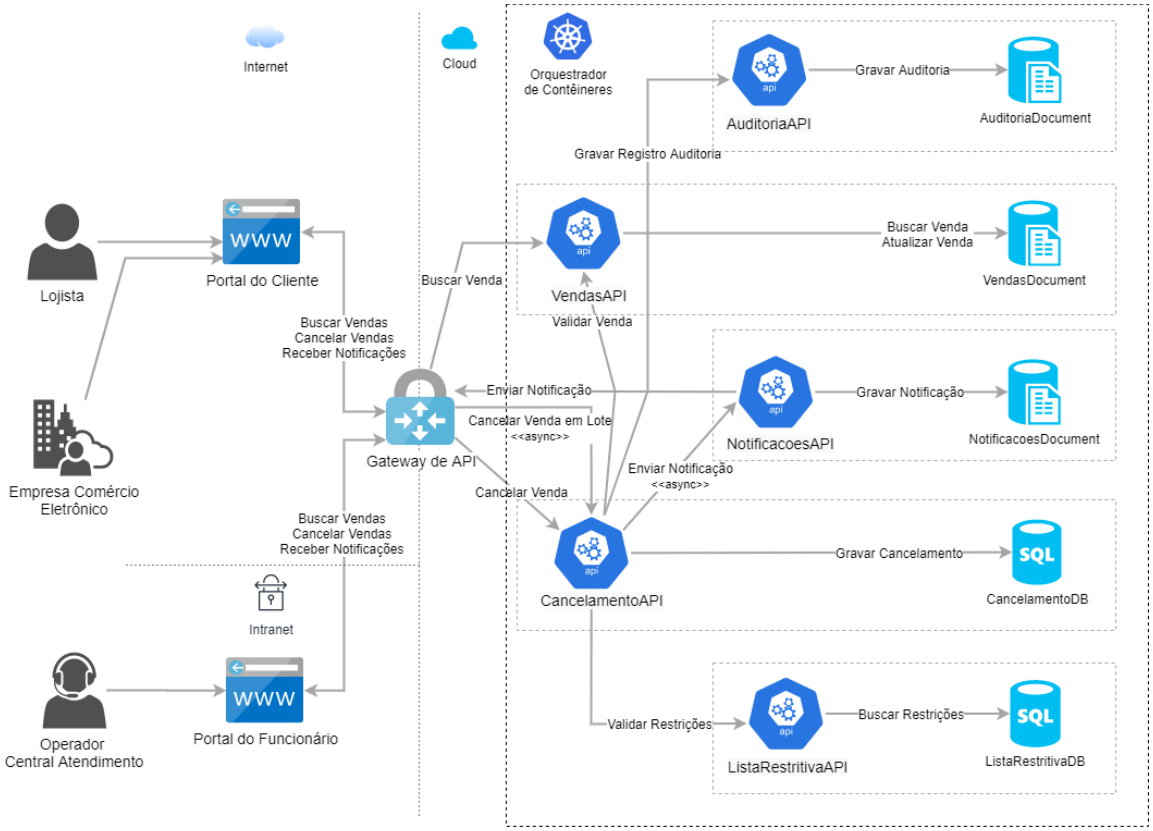

Figura 18 *- Arquitetura da Aplicação - Cenário Proposto*

Fonte: Elaboração própria

O projeto estudo de caso também realizou alterações nas interfaces com os usuários, por meio da reformulação dos portais do cliente e funcionário, otimizando a experiência de navegação; porém, será destacada a inclusão dos componentes diretamente relacionados à arquitetura de microsserviços:

- ***Gateway de API*****:** para centralização de todas as chamadas para as APIs desenvolvidas, bem como autenticação e autorização dos usuários finais ou mesmo de APIs das empresas de comércio eletrônico, eliminando a necessidade do antigo servidor FTP.

- **Orquestrador de Contêineres:** uma vez que todas as novas APIs propostas neste modelo estão contidas em contêineres independentes, o orquestrador possui a responsabilidade de gerenciamento deste novo ambiente, facilitando a escalabilidade.

Cabe ainda mencionar a possibilidade do emprego de tecnologias distintas para o armazenamento de dados: as APIs de auditoria, vendas e notificações passaram a contar com um banco de dados não relacional, permitindo o armazenamento de cada entidade como um documento, obtendo assim mais performance para as consultas realizadas (uma venda consultada irá retornar com todas as informações e parcelas em um único objeto, por exemplo, conforme visto anteriormente na figura 6).

Dando sequência nas ações para atingir o nível 3 do Modelo de Maturidade de Richardson no projeto, foi introduzido também um conjunto padrão de verbos, bem como a capacidade de descoberta nos serviços gerados a partir da decomposição do serviço monolítico original; conforme pode-se observar na figura 19, que exibe à esquerda o cenário atual da aplicação CancelamentoService e à direita o cenário proposto, atingindo assim o nível 3 do MMR.

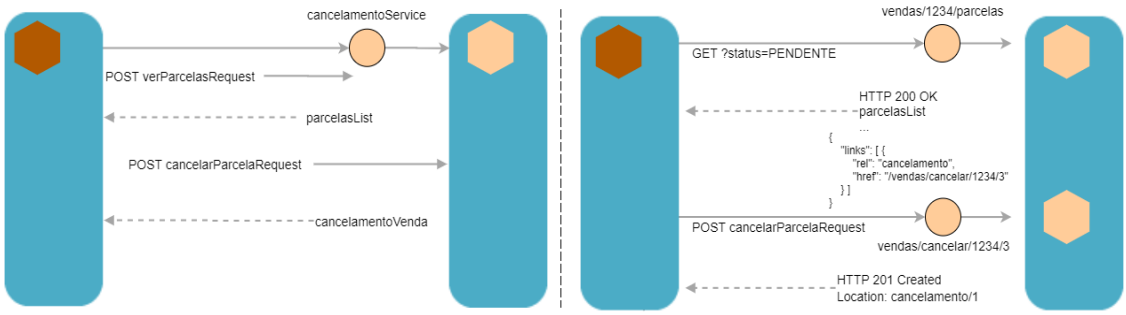

Figura 19 *- Evolução do nível 0 para o nível 3 no Modelo de Maturidade de Richardson*

Fonte: Elaboração própria

Por fim, verifica-se que a nova API para o cancelamento da venda (CancelamentoAPI) é a que possui mais integrações com outras APIs. Apesar de algumas destas chamadas serem assíncronas (estereótipo UML anotado com

“<<*async*>>”) outras possuem integração síncrona, sendo assim a tolerância a falhas um requisito não funcional de grande importância ao projeto.

Durante os testes do projeto foram experimentadas interrupções em determinadas integrações síncronas, algumas delas por esgotar o tempo limite de uma conexão (o chamado “*timeout*”). Observou-se assim a necessidade de adotar e desenvolver programaticamente o padrão de interrupção de circuito, evitando erros em cascata, que ocasionavam em um alto consumo de memória e processamento nos recursos computacionais, refletindo inclusive no custo da infraestrutura em nuvem que foi criada para o projeto.

Atualmente soluções baseadas em “malha de serviços” (do inglês “*service mesh*”) oferecem nativamente recursos para o balanceamento de carga, observabilidade, rastreabilidade e também o padrão de resiliência de interrupção de circuito; sendo assim uma tecnologia de apoio em alguns dos problemas enfrentados no projeto, devolvendo o foco do time de desenvolvimento em agregar valor ao negócio, e não em se preocupar em como a conexão entre os serviços deve funcionar (RED HAT, 2020).

5 Conclusão

Conforme proposto, o presente trabalho apresentou o emprego da Arquitetura de Microsserviços para sistemas de informação baseados em processos de negócio. Sendo este modelo arquitetural um tema frequente e de grande discussão em diversos artigos e publicações de tecnologia em geral, conforme a revisão bibliográfica deste trabalho abordou; é natural inclusive que diversos casos de sucesso e modelos de referência para esta arquitetura sejam considerados em estudos para sua adoção, o que de certa forma também contribui para que ela seja vista muitas vezes como uma solução ideal para quaisquer cenários de projetos.

De fato, a Arquitetura de Microsserviços possui inúmeras vantagens em relação à arquitetura monolítica, porém seus pontos de desvantagens devem ser ponderados em todos os cenários: por introduzir uma maior complexidade ao desenvolvimento, fatores organizacionais como a experiência e até mesmo o tamanho do time de desenvolvimento devem ser considerados antes da sua adoção. Ainda cabe destacar que, requisitos não funcionais como os relacionados à performance e rastreabilidade, são críticos para o sucesso desta arquitetura, sendo necessário também muitas vezes rever modelos de suporte e monitoramento das aplicações da organização; adotando inclusive novas tecnologias para minimizar problemas durante o processo de desenvolvimento, como a malha de serviços, por exemplo.

Entende-se assim que a decisão em se adotar microsserviços deve ser tomada analisando-se também contextos organizacionais, de produto e tecnológicos, não sendo objetivo deste trabalho definir um modelo geral para a sua adoção e/ou migração.

Por meio do estudo de caso foi possível compreender melhor o próprio funcionamento deste padrão arquitetural, bem como suas eventuais fronteiras, vantagens e desvantagens. Ainda foi apresentada a relação direta que microsserviços possui com SOA e a importância de seus princípios de design de serviços, que também são aplicáveis em microsserviços.

Por fim, conclui-se também, por meio do Modelo de Maturidade de Richardson e sua clara divisão em etapas para evolução e/ou avaliação, que há uma ótima

metodologia de suporte para a evolução de arquiteturas monolíticas para microsserviços, permitindo inclusive atingir o nível 3 de maturidade no presente trabalho. Inclusive, considerando-se este modelo, é recomendável que uma possível adoção de microsserviços seja realizada a partir de sistemas monolíticos já existentes, uma vez que este cenário facilita a compreensão do processo de negócio, domínios e suas eventuais fronteiras. Desta forma espera-se que este trabalho e estudo de caso também possam servir de suporte para trabalhos futuros.

Referências Bibliográficas

AMAZON WEB SERVICES. Implementing Microservices on AWS, 2019. **Amazon White Papers.** Disponível em <https://d1.awsstatic.com/whitepapers/microservices-on-aws.pdf>. Acesso em 18 jul. 2020.

ARSOV, Kristijan. Microservices vs. SOA - Is There Any Difference at All? **Medium,** 2017. Disponível em <https://medium.com/@kikchee/microservices-vs-soa-is-there-any-difference-at-all-2a1e3b66e1be>. Acesso em 29 jul. 2020.

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. NBR ISO/IEC 9126-1 **Engenharia de software - Qualidade de produto - Parte 1: Modelo de qualidade**. 2003.

CRUZ, Tadeu. **Sistemas e Métodos & Processos, Administrando Organizações por Meio de Processos de Negócios**. 2ª edição. São Paulo: Editora Atlas SA, 2010.

DUMAS, Marlon et al. **Business process management**. Berlin: Springer-Verlag, 2013.

ERL, Thomas. **SOA Princípios de design de serviços**. Pearson Prentice Hall, São Paulo, 2009.

FIELDING, Roy T.; TAYLOR, Richard N. **Architectural styles and the design of network-based software architectures**. Irvine: University of California, Irvine, 2000.

FOWLER, Martin; LEWIS, James. Microservices a definition of this new architectural term. **Medium**, 2014. Disponível em <http://martinfowler.com/articles/microservices.html>. Acesso em 16 jul. 2020.

FUSCELLA, Arthur Silva. Alcançando A Excelência Do Rest Com Um Modelo De Maturidade Eficiente. **Mundo API**, 2017. Disponível em <https://mundoapi.com.br/destaques/alcancando-a-excelencia-do-rest-com-um-modelo-de-maturidade-eficiente/>. Acesso em 23 jul. 2020.

HEWLETT PACKARD. O que é orquestração de contêiner? 2020. Disponível em <https://www.hpe.com/br/pt/what-is/container-orchestration.html> Acesso em 22 jul. 2020.

JAMSHIDI, Pooyan et al. Microservices: **The journey so far and challenges ahead.** IEEE Software, v. 35, n. 3, p. 24-35, 2018.

JOSUTTIS, Nicolai M. SOA na Prática, **A Arte da Modelagem de Sistemas Distribuídos.** Rio de Janeiro, RJ: Jacaré, 2008.

JUNG, Matthias et al. **Microservices on AWS**. Amazon Web Services, Inc., New York, NY, USA, Tech. Rep, 2016.

JUNIOR, Elemar. Origem E Significado De Rest E Restful. **Eximiaco.tech**, 2019. Disponível em <http://eximiaco.tech/pt/2019/09/13/origem-e-significado-de-rest-e-restful>. Acesso em 21 jul. 2020.

JUSTINO, Yan. microsserviços (e/é) SOA? **Medium**, 2018. Disponível em <https://medium.com/@yanlimaj/microsservi%C3%A7os-e-%C3%A9-soa-1497b34c47ef>. Acesso em 02 maio 2020.

KRAFZIG, Dirk; BANKE, Karl; SLAMA, Dirk. **Enterprise SOA: Service-Oriented Architecture Best Practices**. Indianapolis: Prentice Hall, 2005.

LUBLINSKY, Boris. **Defining SOA as an architectural style**. Estados Unidos: IBM Developers, v. 4, n. 1, p. 1, 2007.

MARTIN, Robert C. **Agile software development: principles, patterns, and practices**. Prentice Hall, 2002.

MICROSOFT. Estilo de arquitetura de microsserviços. **Microsoft.com**, 2019. Disponível em <https://docs.microsoft.com/pt-br/azure/architecture/guide/architecture-styles/microservices>. Acesso em 16 jul. 2020.

NEWMAN, Sam. **Building microservices: designing fine-grained systems**. O'Reilly Media, Inc., 2015.

OPENAPI INITIATIVE et al. OpenAPI specification. **Github**, 2017. Disponível em <https://github.com/OAI/OpenAPI-Specification>. Acesso em 22 jul. 2020.

PRESSMAN, Roger; MAXIM, Bruce. **Engenharia de Software**-8ª Edição. McGraw Hill Brasil, 2016.

RED HAT. Containers: o que é docker? **RedHat.com**, 2020. Disponível em <https://www.redhat.com/pt-br/topics/containers/what-is-docker>. Acesso em 22 jul. 2020.

RED HAT. Interface De Programação De Aplicações. **RedHat.com**, 2020. Disponível em <https://www.redhat.com/pt-br/topics/api/what-are-application-programming-interfaces>. Acesso em 20 jul. 2020.

RED HAT. O que é service mesh? **RedHat.com**, 2020. Disponível em <https://www.redhat.com/pt-br/topics/microservices/what-is-a-service-mesh >. Acesso em 02 set. 2020.

RICHARDS, Mark. **Microservices vs. service-oriented architecture**. O'Reilly Media, 2015.

RICHARDSON, Leonard. Richardson Maturity Model. **MartinFowler.com**, 2010. Disponível em <https://martinfowler.com/articles/richardsonMaturityModel.html>. Acesso em 22 jul. 2020.

SABBAGH, Rafael. **Scrum: Gestão ágil para projetos de sucesso**. Editora Casa do Código, 2014.

SCHWABER, Ken; BEEDLE, Mike. **Agile software development with Scrum**. Upper Saddle River: Prentice Hall, 2002.

Workflow Management Coalition. “The Workflow Management Coalition Specification - Terminology & Glossary”. Document Number WFMC-TC-1011, Feb 1999.