



SERVIÇO PÚBLICO FEDERAL

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE GOIÁS

PRÓ-REITORIA DE ENSINO

CÂMPUS GOIÂNIA

BACHARELADO EM SISTEMAS DE INFORMAÇÃO

MAURO PIRES MOREIRA NETO

VINICIUS DA SILVA E SOUSA AUGUSTO

Padrões para produção de aplicações utilizando Microsserviços

Goiânia, 2021.

SERVIÇO PÚBLICO FEDERAL
INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE GOIÁS
PRÓ-REITORIA DE ENSINO
CÂMPUS GOIÂNIA
BACHARELADO EM SISTEMAS DE INFORMAÇÃO

MAURO PIRES MOREIRA NETO
VINICIUS DA SILVA E SOUSA AUGUSTO

Padrões para produção de aplicações utilizando Microsserviços

Trabalho de Conclusão de Curso apresentado à Coordenação do Curso de Bacharelado em Sistemas de informação como requisito parcial para obtenção do título de Bacharel em Sistemas de informação.

Orientador: Prof. Dr. Raphael de Aquino Gomes

Goiânia, 2021.

M838p Moreira Neto, Mauro Pires.

Padrões para produção de aplicações utilizando microsserviços / Mauro Pires Moreira Neto; Vinicius da Silva e Sousa Augusto. – Goiânia : Instituto Federal de Educação, Ciência e Tecnologia de Goiás, 2021.
76f.

Orientador: Prof. Dr. Raphael de Aquino Gomes.

TCC (Trabalho de Conclusão de Curso) – Curso de Bacharelado em Sistemas de Informação, Instituto Federal de Educação, Ciência e Tecnologia de Goiás.

1. Arquitetura de software. 2. Microsserviços. I. Augusto, Vinicius da Silva e Sousa. II. Gomes, Raphael de Aquino (orientador). III. Instituto Federal de Educação, Ciência e Tecnologia de Goiás. IV. Título.

CDD 005.1

Ficha catalográfica elaborada pela bibliotecária Lana Cristina Dias Oliveira CRB1/ 2.631
Biblioteca Professor Jorge Félix de Souza,
Instituto Federal de Educação, Ciência e Tecnologia de Goiás, Câmpus Goiânia.

**TERMO DE AUTORIZAÇÃO PARA DISPONIBILIZAÇÃO
NO REPOSITÓRIO DIGITAL DO IFG - ReDi IFG**

Com base no disposto na Lei Federal nº 9.610/98, AUTORIZO o Instituto Federal de Educação, Ciência e Tecnologia de Goiás, a disponibilizar gratuitamente o documento no Repositório Digital (ReDi IFG), sem resarcimento de direitos autorais, conforme permissão assinada abaixo, em formato digital para fins de leitura, download e impressão, a título de divulgação da produção técnico-científica no IFG.

Identificação da Produção Técnico-Científica

- | | |
|--|---|
| <input type="checkbox"/> Tese | <input type="checkbox"/> Artigo Científico |
| <input type="checkbox"/> Dissertação | <input type="checkbox"/> Capítulo de Livro |
| <input type="checkbox"/> Monografia – Especialização | <input type="checkbox"/> Livro |
| <input checked="" type="checkbox"/> TCC - Graduação | <input type="checkbox"/> Trabalho Apresentado em Evento |
| <input type="checkbox"/> Produto Técnico e Educacional - Tipo: _____ | |

Nome Completo do Autor: *Mauro Ribeiro Moreira Neto / Vinícius da Silveira Augusto*
Matrícula: *20151011090162 / 20151011090375*

Título do Trabalho: *Padrões para produção de aplicações utilizando Microserviços*

Autorização - Marque uma das opções

1. Autorizo disponibilizar meu trabalho no Repositório Digital do IFG (acesso aberto);
2. Autorizo disponibilizar meu trabalho no Repositório Digital do IFG somente após a data ____/____/____ (Embargo);
3. Não autorizo disponibilizar meu trabalho no Repositório Digital do IFG (acesso restrito).

Ao indicar a opção 2 ou 3, marque a justificativa:

- O documento está sujeito a registro de patente.
 O documento pode vir a ser publicado como livro, capítulo de livro ou artigo.
 Outra justificativa: _____

DECLARAÇÃO DE DISTRIBUIÇÃO NÃO-EXCLUSIVA

O/A referido/a autor/a declara que:

- i. o documento é seu trabalho original, detém os direitos autorais da produção técnico-científica e não infringe os direitos de qualquer outra pessoa ou entidade;
- ii. obteve autorização de quaisquer materiais inclusos no documento do qual não detém os direitos de autor/a, para conceder ao Instituto Federal de Educação, Ciência e Tecnologia de Goiás os direitos requeridos e que este material cujos direitos autorais são de terceiros, estão claramente identificados e reconhecidos no texto ou conteúdo do documento entregue;
- iii. cumpriu quaisquer obrigações exigidas por contrato ou acordo, caso o documento entregue seja baseado em trabalho financiado ou apoiado por outra instituição que não o Instituto Federal de Educação, Ciência e Tecnologia de Goiás.

Goiânia _____, 10/03/21
Local _____
Data _____

Vinícius da Silveira Augusto / Mauro Ribeiro Moreira Neto
Assinatura do Autor e/ou Detentor dos Direitos Autorais

SERVIÇO PÚBLICO FEDERAL
INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE GOIÁS
PRÓ-REITORIA DE ENSINO
CÂMPUS GOIÂNIA
BACHARELADO EM SISTEMAS DE INFORMAÇÃO

MAURO PIRES MOREIRA NETO
VINICIUS DA SILVA E SOUSA AUGUSTO

Padrões para produção de aplicações utilizando Microsserviços

Trabalho de Conclusão de Curso defendido no Curso de Bacharelado em Sistemas de informação como requisito parcial para obtenção do título de Bacharel em Sistemas de informação, aprovada em 10 de Março de 2021, pela Banca Examinadora constituída pelos professores:

Raphael de Aquino Gomes

Prof. Dr. Raphael de Aquino Gomes
Departamento IV - IFG / Câmpus Goiânia
Presidente da Banca

Gustavo Cipriano Mota Sousa

Prof. Dr. Gustavo Cipriano Mota Sousa
Departamento IV - IFG / Câmpus Goiânia

Eduardo Noronha de Andrade Freitas

Prof. Dr. Eduardo Noronha de Andrade Freitas
Departamento IV - IFG / Câmpus Goiânia

Dedicatória

Eu Mauro, dedico este trabalho aos meus familiares e amigos, que me ajudaram a continuar independente de qualquer situação e que ficaram ao meu lado em todo momento, e ao nosso orientador que nos motivou a continuar mesmo durante crises, sempre nos apoiando e ensinando nos momentos mais difíceis.

Eu Vinicius, dedico este trabalho a toda minha família, que em todos os momentos, desde o começo do curso até agora, têm me apoiado e me dado todo o suporte necessário para eu seguir em frente e trilhar meus sonhos.

Agradecimentos

Eu Mauro, agradeço a todos os meus familiares por terem me apoiado incondicionalmente durante o caminho percorrido até aqui, principalmente aos meus pais Carlos e Kátia que me deram todo o apoio que precisei para conseguir concluir este trabalho. Ao meu parceiro neste trabalho, Vinícius, ao qual sem ele não teria sido possível concluir o trabalho, apesar de todas as dificuldades. Ao professor e orientador Raphael de Aquino que nos mostrou tudo que precisávamos para seguir em frente, com uma incrível dedicação e paciência para nos mostrar o caminho além de qualquer dificuldade. Obrigado a todos envolvidos pelo apoio e auxílio nesta jornada.

Eu Vinicius, agradeço a toda a minha família por desde sempre apoiar e acreditar em mim. Ao meu irmão Leonardo por sempre me instigar a correr atrás dos meus sonhos e sempre buscar conhecimento. Meus pais Edner e Luciete que sempre foram os pilares de sustentação e como faróis na noite, abrindo caminho para que os filhos sempre pudessem seguir com seus sonhos e desejos através de oportunidades. À minha avó Almerinda, uma pessoa que sempre cuidou e se preocupou comigo, além de sempre ser uma pessoa enérgica e portadora de palavras de conforto. À minha avó Lourença, que mesmo sendo uma pessoa bem simples, conseguiu me transmitir um conhecimento que nenhuma faculdade ou livro consegue ensinar, conhecimento aquele que não é obtido em textos, mas sim, na vida. À minha sogra Sandra, por em momentos bons e ruins sempre me aconselhar e orientar, sempre com uma postura sóbria, e por fim, à minha esposa Amanda, por todos os momentos de apoio e de sempre me motivar a seguir em frente apesar de tudo, sempre acreditando nas coisas e me incentivando em tudo que eu penso e faço, sendo amiga e companheira em todos os momentos. Todos vocês são pessoas incríveis e aqui eu deixo meu agradecimento.

O que sabemos, saber que o sabemos. Aquilo que não sabemos, saber que não o sabemos: eis o verdadeiro saber.

Confúcio,
551 a.C. - 479 a.C..

Resumo

Título: Padrões para produção de aplicações utilizando Microsserviços

Autores: Mauro Pires Moreira Neto e Vinicius da Silva e Sousa Augusto

Orientador: Dr. Raphael de Aquino Gomes

Este trabalho parte do comparativo entre algumas Arquiteturas de Software, sendo elas a Arquitetura Monolítica, a Arquitetura Orientada a Serviços e a Arquitetura de Microsserviços, dando enfoque nesta última. É realizado um detalhamento mais profundo desta Arquitetura, bem como algumas ferramentas para o desenvolvimento da mesma, e em seguida, realizado um estudo buscando levantar e catalogar os padrões de Microsserviços mais comuns, sendo estes imediatamente descritos com maiores detalhes, e posteriormente, tendo sua utilização na construção de uma aplicação com a Arquitetura de Microsserviços, dentro da nuvem da Amazon Web Services. Também foi realizado o desenvolvimento de uma Arquitetura Monolítica dentro da nuvem, sendo submetidas a avaliações, que mostram um ganho significativo em qualidade e desempenho ao utilizar padrões de Microsserviços, apesar de tal Arquitetura demandar um custo maior em seu desenvolvimento. Por fim, é sugerido trabalhos futuros, como melhorias que podem reduzir o custo da Arquitetura, desenvolver a Arquitetura de Microsserviços com padrões sem utilizar a nuvem e um estudo mais detalhado de cada padrão.

Palavras-chave

Microsserviços, Patterns, Amazon Web Services, Arquitetura de software

Abstract

Title: Patterns for development of applications using Microservices

Authors: Mauro Pires Moreira Neto and Vinicius da Silva e Sousa Augusto

Advisor: Dr. Raphael de Aquino Gomes

This work is the comparison between some software architectures, being the monolithic architecture, the service-oriented architecture and the Microservice architecture, focusing on the Microservice architecture. So that a more in-depth detailing of this architecture is made, as well as some tools for its development, and then a study is carried out seeking to raise and catalog the most common microservice patterns, these patterns being immediately previous in greater detail and having its use in the construction of an application with microservices architecture, within the cloud of Amazon Web Services. A monolithic architecture was also developed within the cloud, so that both can be submitted to evaluations, which show a significant gain in using microservice patterns, despite the fact that such architecture demands a higher cost in its development. Finally, we will describe future work as improvements that can reduce this cost, develop a microservice architecture with patterns without using the cloud, and a more detailed study of each pattern.

Keywords

Microservices, Patterns, Amazon Web Services, Software architecture

Lista de Figuras

2.1	Arquitetura Monolítica	20
2.2	Arquitetura SOA	21
2.3	Arquitetura de Microsserviços	22
2.4	Diferentes dimensões para escalabilidade de uma aplicação.	27
3.1	Api Gateway fazendo o direcionamento das requisições para os micro-serviços	31
3.2	Controle de acesso e requisições utilizando um Token	32
3.3	Banco de dados por serviço	32
4.1	Diagrama de caso de uso	36
4.2	Arquitetura Monolítica implantada na AWS	40
4.3	Arquitetura de Microsserviços implantada na AWS	42
5.1	Teste de carga	46
5.2	Teste de performance	47

Lista de Tabelas

2.1	Tabela comparativa entre Microsserviços e SOA	26
5.1	Tabela de custos Monolítico	49
5.2	Tabela de custos Microsserviço	50

Lista de Abreviaturas e Siglas

AWS	Amazon Web Services
EC2	Amazon Elastic Compute Cloud
ECR	Amazon Elastic Container Registry
ECS	Amazon Elastic Container Service
ELB	Amazon Elastic Load Balancing
JWT	JSON Web Tokens
MS	Microsserviços
RDS	Amazon Relational Database Service
RPC	Remote Procedure Call
SO	Sistema Operacional
SOA	Service-oriented architecture

Sumário

LISTA DE FIGURAS	10
LISTA DE TABELAS	11
LISTA DE ABREVIATURAS E SIGLAS	12
1 INTRODUÇÃO	15
1.1 Objetivos	17
1.2 Metodologia	17
1.3 Resultados Esperados	18
1.4 Organização do Trabalho	18
2 PADRÃO ARQUITETURAL DE SOFTWARE	19
2.1 Padrão Arquitetural Monolítico	19
2.2 Arquitetura Orientada a Serviços	20
2.3 Padrão Arquitetural Orientado a Microsserviços	22
2.3.1 Ferramentas Utilizadas no Desenvolvimento de Microsserviços	23
2.3.1.1 Spring - JAVA (WEISSMANN, 2014)	23
2.3.1.2 Docker (ANDERSON, 2015)	23
2.3.1.3 Computação em Nuvem (<i>Cloud Computing</i>)	24
2.4 Análise das Arquiteturas	25
2.5 Considerações Finais	28
3 PADRÕES PARA O DESENVOLVIMENTO DE UMA ARQUITETURA VOL-	
TADA A MICROSSERVIÇOS	29
3.1 Integração	30
3.1.1 <i>API Gateway</i>	30
3.1.2 <i>Backends de API</i>	31
3.1.3 <i>Token de Acesso (Access Token)</i>	31
3.2 Banco de dados	32
3.3 Serviços	33
3.3.1 Disjuntor (<i>Circuit Breaker</i>)	33
3.3.2 Padrões de Descoberta (<i>Discovery Patterns</i>)	33
3.3.3 Descoberta do Lado do Cliente (<i>Client-Side Discover</i>)	33
3.3.4 Descoberta do Lado do Servidor (<i>Server-Side Discover</i>)	34
3.3.5 Registro de Serviço (<i>Service Registry</i>)	34
3.4 Considerações Finais	34

4 ESTUDO DE CASO SOBRE A APLICAÇÃO DE PADRÕES DE MICROSSERVIÇOS	35
4.1 Caracterização da aplicação	36
4.2 Metodologia de implantação	37
4.3 Serviços utilizados para compor a aplicação	37
4.3.1 Serviços do AWS utilizados na construção das arquiteturas	38
4.4 Arquitetura da Aplicação monolítica	39
4.5 Transformação da Aplicação Monolítica para Microsserviços com seus padrões	40
4.6 Considerações Finais	43
5 AVALIAÇÃO	44
5.1 Avaliação da ferramenta dos testes	45
5.2 Seleção dos testes realizados	45
5.2.1 Teste de Carga	45
5.2.2 Teste de performance	46
5.3 Avaliação sobre tolerância a falhas	48
5.4 Avaliação de custo das arquiteturas	48
5.5 Considerações Finais	50
6 CONSIDERAÇÕES FINAIS E TRABALHOS FUTUROS	53
REFERÊNCIAS BIBLIOGRÁFICAS	55

CAPÍTULO 1

Introdução

Com o rápido avanço tecnológico dos últimos anos, em que uma tecnologia é desenvolvida e rapidamente está sendo substituída por outra mais moderna ou eficiente, teve como consequência o impulso de vários setores da sociedade, como o da Computação. Os equipamentos computacionais que eram considerados os melhores e mais avançados há algumas décadas atrás, hoje, além de serem uma lembrança dos primórdios da tecnologia computacional, são bastante inferiores se comparados aos computadores modernos, como o *Raspberry Pi*, muito menor e com capacidade de processamento bem mais elevado que tais equipamentos.

A crescente melhoria na tecnologia permitiu também que a área de Desenvolvimento de Sistemas acompanhasse este crescimento, incitando a possibilidade de sistemas cada vez melhores, com mais funcionalidades e integrações com outros sistemas. Isso, por sua vez acarretou em um aumento no nível de complexidade dos mesmos, que por consequência geraram grandes problemas na sua manutenção e na adição de novas funcionalidades.

Tais problemas podem ser relacionados ao fato de que a maioria das aplicações são construídas de forma interligada, em que vários componentes geram a aplicação em si, sendo necessário para o seu funcionamento que todas as partes estejam funcionando corretamente. Essa forma de construção de sistemas é tradicionalmente chamada de Arquitetura Monolítica. Isso ocorre por ser uma forma simples e rápida de construção de sistemas, ou seja, uma forma onde toda a aplicação é construída junta e interligada (por isso a referência a um monólito), sendo que questões de escalabilidade e de desatribuição são tratadas de forma única, sem ser possível realizar especificidades em apenas uma parte do código.

Entretanto, ao pensarmos nos sistemas modernos onde há a necessidade de uma boa escalabilidade, realização de manutenções e melhorias, além existirem sistemas cada vez maiores e com diversas funcionalidades, integrações, e mais recentemente a necessidade de adicionar áreas de Inteligência artificial e Inteligência de Negócios, vemos que tal Arquitetura não é suficientemente adequada para tais aplicações devido a sua construção solida.

Assim, pode-se pensar que, unindo tudo em um único código composto por vários módulos, gera-se os problemas descritos anteriormente. Há a possibilidade de fragmentar o sistema em pequenos pacotes de código, que por sua vez podem ser totalmente independentes nas suas funções, podendo trazer benefícios a fim de solucionar os problemas descritos.

A metodologia da Arquitetura de Microsserviços (MS) oferece esta possibilidade. Segundo Newman (2015)"Os Microsserviços são pequenos serviços autônomos que funcionam juntos". E, conforme Richardson (b), os Microsserviços são, "altamente sustentáveis e testáveis, fracamente acoplados, implementados de forma independente, organizados em torno de recursos de negócios".

Pode-se dizer que a Arquitetura de Microsserviços consiste na fragmentação em pequenos pacotes de código completos e funcionais. Tais pacotes podem ou não serem associados de forma a gerar grandes sistemas complexos, mas que ao mesmo tempo permita uma tolerância a falhas. Essa tolerância se deve ao fato de que se um desses módulos deixar de funcionar, a única parte afetada será a do módulo em questão. Além disso, permite a inserção de novos componentes sem necessitar a refatoração do código todo, buscando interligar o módulo novo ao restante do código, bastando apenas associar as partes novas aos serviços já existentes. Tudo isso, simplifica a manutenibilidade da aplicação, sendo executada apenas no serviço em que se deseja esse fim, além de possibilitar que esses pacotes sejam distribuídos, gerando redundâncias e escalabilidade conforme a necessidade de uso de cada módulo específico.

Entretanto, para que essa vantagem ocorra é necessária a utilização de padrões de Microsserviços. Conforme (ALEXANDER, 1977) "Cada padrão é uma regra de três partes, uma que expressa uma relação entre um determinado contexto, um problema e uma solução". Assim, os padrões visam resolver os problemas descritos anteriormente, propondo formas de se implementar soluções, buscando gerar um sistema mais robusto dentro da ideia de Microsserviços.

Padrões também visam gerar uma maior similaridade para as pessoas que estão trabalhando com eles, servindo como um manual de instruções para buscar os locais corretos onde atuar, bem como auxiliando a resolução de erros, já que é possível ter a noção de como deveria ser o funcionamento da parte específica do software, além de utilizar técnicas já executadas anteriormente para a resolução do problema encontrado.

Na questão Arquitetural, existem outros modelos arquiteturais que também propõem a divisão do sistema em partes, a exemplo da Arquiteturas Orientadas a Serviço (SOA), que por sua vez sugere dividir o sistema em Serviços que são interligados, a fim de gerar uma aplicação maior. Alguns autores sugerem que SOA e MS são intimamente ligados, como para Taibi2018 que diz que "Os Microsserviços surgiram como uma variante do SOA". É inegável que ambos compartilham de muitas similaridades em

suas construções, porém conforme aprofunda-se sobre o assunto observa-se que cada Arquitetura tem suas peculiaridades, e afirmar que os Microsserviços são apenas uma variante de SOA pode não fazer tanto sentido.

Assim, neste trabalho será realizada a busca e o estudo de padrões de Microsserviços, visando mostrar como tais padrões, ao serem aplicados, geram utilizações melhores do que quando não aplicados. O estudo busca demonstrar as Arquiteturas de Software mais comumente utilizadas, dando um foco específico para Arquitetura de Microsserviços, mostrando algumas ferramentas para a construção de tal Arquitetura. Em seguida, será feita uma análise comparativa entre tais Arquiteturas, e posteriormente, serão demonstrados os padrões de Microsserviços com a utilização de padrões.

1.1 Objetivos

É proposto neste trabalho demonstrar padrões de Microsserviços e como sua utilização pode ser uma excelente alternativa para se adicionar na construção de softwares utilizando os mesmos; encontrar e catalogar padrões de Microsserviços existentes e suas funções; exemplificar ferramentas que facilitem o desenvolvimento de Microsserviços e a aplicação de padrões; investigar as vantagens em se desenvolver uma aplicação utilizando padrões de Microsserviços; comparar a implementação e implantação de uma aplicação utilizando o padrão Arquitetural Monolítico baseado em padrões de Microsserviços.

1.2 Metodologia

A metodologia utilizada para o projeto é a pesquisa exploratória. Buscando a familiarização dos discentes com o tema, foi realizada uma pesquisa bibliográfica, visando encontrar métodos, projetos, trabalhos práticos e ferramentas sobre o tema relacionado. Após feito um catálogo de referências e materiais relacionados ao assunto proposto, foi realizado o levantamento do material e uma análise sobre padrões de projeto, a fim de dividir os mesmos em padrões de SOA, padrões para Microsserviços e padrões que eram de SOA mas poderiam ser aplicáveis ao Microsserviços.

Após essa análise foi realizada a criação de uma catalogação dos padrões de Arquiteturas. O enfoque dessa tabela foram os padrões de Microsserviços, onde foi constatado a necessidade de buscar mais padrões, visando catalogar aqueles que mais se repetiam. Realizou-se também uma análise das ferramentas encontradas para o desenvolvimento de Microsserviços. Paralelamente, estudou-se as aplicações disponíveis na internet que eram construídas de forma Monolítica e que poderiam ser convertidas para Microsserviços utilizando os padrões encontrados e as ferramentas propostas. Feito isso, foi iniciada a fase de consolidação da parte escrita do trabalho.

1.3 Resultados Esperados

Como resultados esperados existe a necessidade de que sejam levantados os principais padrões para criação de aplicações utilizando Microsserviços, e em seguida a criação de um catálogo com esses padrões para serem utilizados na construção de tais aplicações, além do levantamento de ferramentas para o desenvolvimento de Microsserviços. Em seguida, após obter os conhecimentos descritos anteriormente, seja possível converter um sistema desenvolvido em uma Arquitetura Monolítica para um sistema com a Arquitetura de Microsserviços utilizando os padrões.

Também espera-se ser possível demonstrar que, com a utilização de testes em ambas Arquiteturas, a de Microsserviços se demonstre bem mais vantajosa comparativamente a Monolítica, principalmente nos quesitos de escalabilidade e de mantinabilidade da aplicação.

1.4 Organização do Trabalho

Este trabalho está dividido em cinco capítulos. O capítulo 2 serve como contextualização de algumas Arquiteturas comumente utilizadas no desenvolvimento de software, com enfoque para a Arquitetura de Microsserviços, demonstrando inclusive ferramentas para o desenvolvimento da mesma. O Capítulo 3 trata dos principais padrões utilizados no desenvolvimento de uma Arquitetura de Microsserviços, sendo detalhando aqueles que mais se repetiam no catálogo que foi criado dentro do desenvolvimento do capítulo. O Capítulo 4 se refere à demonstração arquitetural de uma aplicação Monolítica e em seguida como converter a mesma para uma aplicação utilizando Microsserviços, além de demonstrar a lógica funcional interna de ambas Arquiteturas. O Capítulo 5 é a parte onde foi realizada testes nas Arquiteturas demonstradas no Capítulo 4, demonstrando uma comparação direta entre ambas Arquiteturas. E por fim o capítulo 6 trazendo algumas considerações finais que foram elencadas após o estudo, bem como propostas de trabalhos futuros correlatos ao trabalho desenvolvido.

CAPÍTULO 2

Padrão Arquitetural de Software

Neste capítulo será discutido três padrões Arquiteturais de software: Arquitetura Monolítica, Arquitetura Orientada a Serviços e Arquitetura Orientada a Microsserviços, visando expor características de cada uma, como estas propõem o desenvolvimento de software e a problematização de sua aplicabilidade.

Visando as qualidades e defeitos destas Arquiteturas no aspecto de desenvolvimento de novos *Softwares*, será demonstrado a necessidade de estudar sobre as mesmas. O fato da necessidade de novas ferramentas para o avanço de tecnologias está diretamente ligado a codificação, e em muitas das vezes, é o principal motivo de buscar uma visão mais aprofundada destas Arquiteturas. Sendo assim, será analisado os padrões Arquiteturais mais utilizados atualmente. Ao final desta análise, será realizado um comparativo entre estes padrões Arquiteturais para mostrar o porquê utilizar cada um deles.

2.1 Padrão Arquitetural Monolítico

A Arquitetura Monolítica possui certas linguagens que auxiliam o código na quebra de funções para permitir uma melhora na complexidade dos sistemas, porém, está consiste em uma aplicação formada por vários módulos que, mesmo agindo separadamente, continuam ligados, transformando o conjunto de módulos em um único sistema. Há diversos sistemas que utilizam esta aplicação, devido à facilidade de criar tudo de maneira única, interligando as funcionalidades do sistema. Porém, a Arquitetura Monolítica tem grandes problemas advindos de sua própria filosofia de construção de *software*, como citado por Batista (2018): "... problemas de escalabilidade, agregação de tecnologias e uma enorme demora no aculturamento para novos funcionários...". Há ainda outros problemas:

- O aumento na complexidade e tamanho do código ao longo do tempo;
- Alta dependência de componentes de código;
- Falta de flexibilidade;
- Dificuldade para colocar alterações em produção.

Tais problemas advindos destas circunstâncias podem e muitas vezes vão dificultar na maior parte do desenvolvimento, em troca de um pequeno sentimento de conforto para a criação do código, como citado anteriormente.

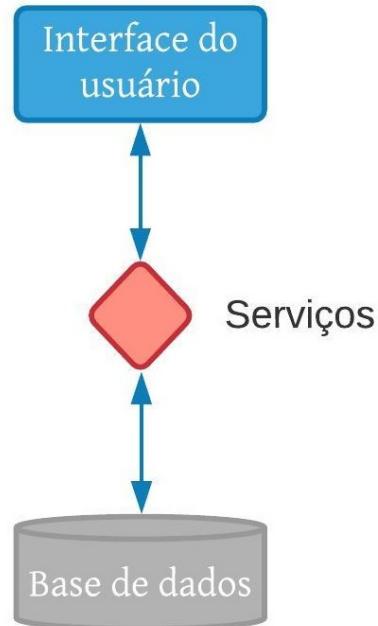


Figura 2.1: Arquitetura Monolítica

Fonte: Elaborada pelos autores.

A Figura 2.1 retrata a topologia de uma Arquitetura Monolítica, em que fica evidente a relação dos agrupamentos que são necessários para o funcionamento de uma aplicação que é feita usando esta Arquitetura, a complexidade das seções que são necessárias para o funcionamento e a relação entre elas, e acima de tudo mostra o quanto frágil é a Arquitetura, pois qualquer indício de perda de dados ou falha de processamento em qualquer etapa acarretará em uma pane na aplicação o que resultará em paralisação total do sistema.

2.2 Arquitetura Orientada a Serviços

Agora, quando se fala de Arquitetura Orientada a Serviços (SOA), muitos pensam que é uma Arquitetura em si, que possui um corpo modelo (metodologia) para o desenvolvimento, ou é um serviço em si que já está pronto para uso. Porém, é muito pelo contrário. De acordo com Sampaio (2006), a Arquitetura Orientada a Serviços (SOA) consiste em um "Paradigma de desenvolvimento de aplicações cujo objetivo é criar módulos funcionais chamados de serviços, com baixo acoplamento e permitindo a

reutilização do código.", assim, um sistema que divide suas funcionalidades em módulos permitirá uma melhor manutenibilidade do código e do sistema, além de permitir a reutilização do código com maior facilidade como podemos observar na Figura a seguir.

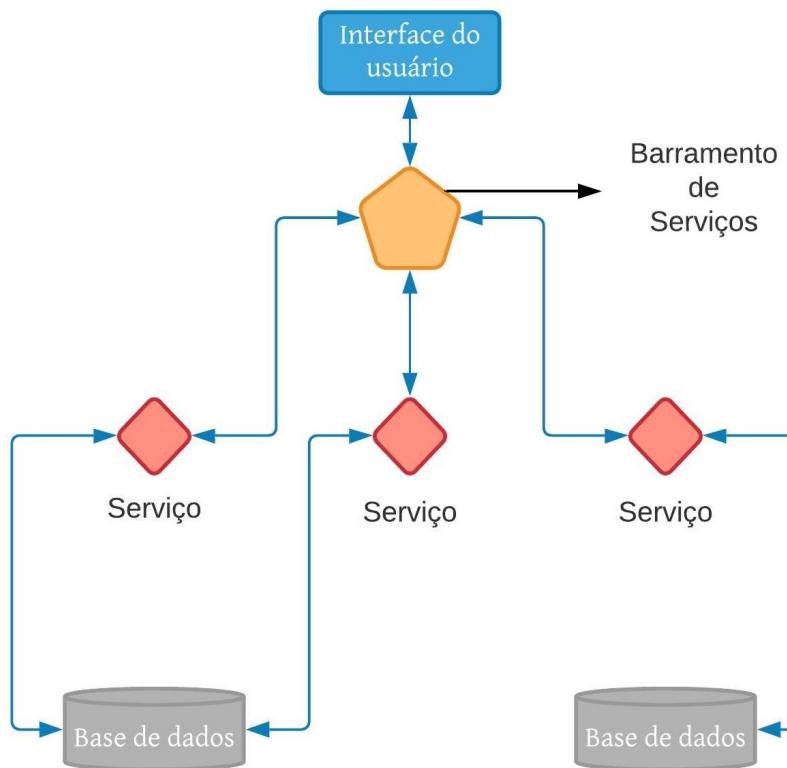


Figura 2.2: Arquitetura SOA

Fonte: Elaborada pelos autores.

Como observado na Figura 2.2 e na descrição dada antes, pode-se observar que a separação dos métodos em serviços “independentes”, se propõe a entregar uma melhor manutenibilidade do código-chave e dos métodos filhos que se tornam dependentes diretos destes serviços. Porém, é visível a melhora do conceito e da segurança para a aplicação caso ocorra falhas em suas partes, sendo assim permitindo a funcionalidade de algumas partes mesmo que outras estejam paradas.

Porém, não se pode usar a Arquitetura Orienta a Serviços (SOA) em qualquer sistema. Ainda de acordo com Sampaio (2006): "Não podemos desenvolver um sistema apenas pensando em serviços, pois corremos o risco de introduzir redundância descontrolada no sistema", fato este que pode gerar um aumento na carga para com o sistema no sentido do processamento, ou manter diversos serviços redundantes, tornando-se difícil o que deveria ser um processo mais prático e seguro de desenvolver novos *softwares*. Ou seja, é necessária uma avaliação mais aprofundada da necessidade destes serviços e da real presença deles no sistema, criando e mantendo apenas aqueles serviços essenciais, e

toda vez que surgir a necessidade de um novo, rever o conceito dos já existentes para não criar os problemas descritos anteriormente.

2.3 Padrão Arquitetural Orientado a Microsserviços

Muitos definem Microsserviços como uma Arquitetura sem um sentido prévio, apenas um conceito genérico para falar sobre serviços independentes que assim definem a Arquitetura, como Batista (2018) diz: "Microsserviços é a separação de elementos de funcionalidade colocados em serviços separados, dessa forma tornando-os totalmente autônomos e totalmente independentes", sendo assim, os Microsserviços se caracterizam como uma prática bem semelhante a Arquitetura Orientada a Serviços (SOA), porém, com certas singularidades, como o fato de que em Microsserviços, cada funcionalidade é separada em serviços bem definidos que podem ou não complementar outra funcionalidade, enquanto a SOA define serviços como módulos funcionais.

A Figura 2.3 abaixo representa uma topologia de uma Arquitetura baseado em Microsserviços, a qual demonstra suas singularidades em relação a topologia da Arquitetura anterior (Arquitetura Orientada a Serviços).

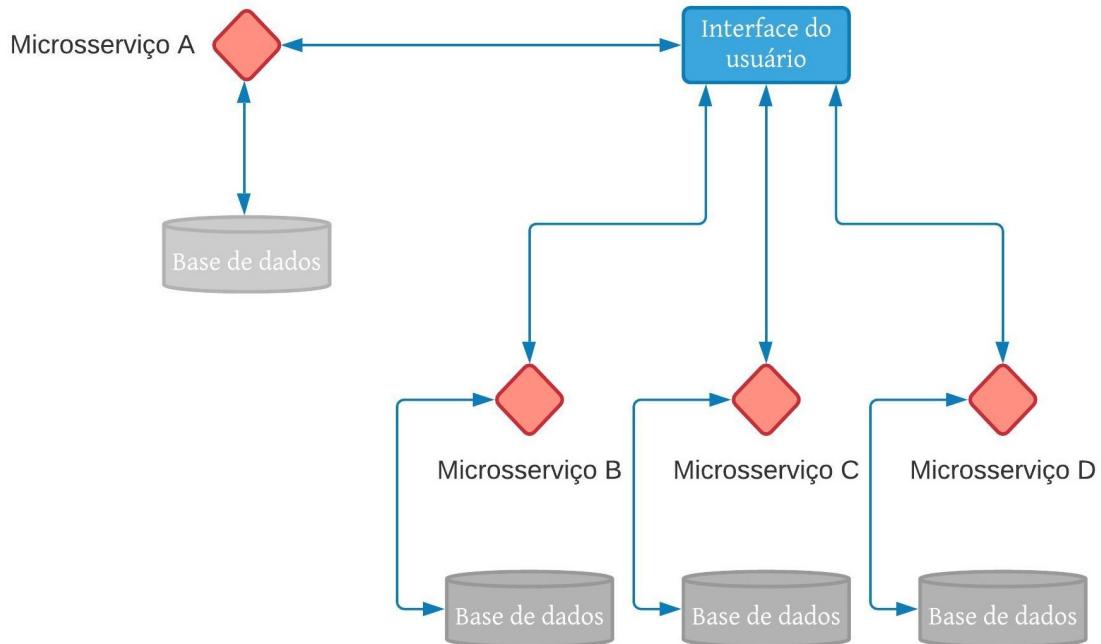


Figura 2.3: Arquitetura de Microsserviços

Fonte: Elaborada pelos autores.

Entretanto, conforme a pesquisa realizada por Taibi *et al.* (2017), existem algumas dificuldades na utilização dos Microsserviços, como a necessidade de exigir desenvolvedores muito experientes e qualificados, devido à grande dificuldade na compreensão

da Arquitetura de Microsserviços e no conhecimento sobre as práticas aplicadas nos mesmos, como também é visível na topologia abaixo, a qual diversos serviços podem interagir entre si, porém a necessidade de saber como usá-los muitas vezes é um empecilho para várias pessoas/empresas.

2.3.1 Ferramentas Utilizadas no Desenvolvimento de Microsserviços

Quando tratamos do desenvolvimento dos padrões Arquiteturais, recorre-se ao uso de ferramentas que auxiliam no desenvolvimento, onde será abordado seu uso e o motivo de ter sido escolhido, assim também, dando uma amostra da eficácia de Frameworks no desenvolvimento de tais padrões.

2.3.1.1 Spring - JAVA (WEISSMANN, 2014)

Durante a revisão de ferramentas utilizadas para o desenvolvimento de Microsserviços, a que mais se mostrou frequente foi o Framework Spring utilizando a linguagem Java. De acordo com (WEISSMANN, 2014) "*Spring Framework* é um *framework* voltado para o desenvolvimento de aplicações corporativas para a plataforma Java, baseado nos conceitos de inversão de controle e injeção de dependências. Esta é a descrição padrão que foi encontrado na grande maioria dos materiais sobre o assunto.

Juntando o fato dele ser predominantemente uma das ferramentas mais utilizadas quando se trata de serviços e o fato de o Spring ser "...modular permitindo que você use apenas as partes que precisa, sem ter que trazer o resto" (JOHNSON *et al.*, 2004) traz à tona a importância da ferramenta para diversos desenvolvedores e empresas. Além disso, tal modularização e a possibilidade de se utilizar apenas o necessário, juntamente com a linguagem JAVA, que por sua vez também fornece um bom suporte para a modularização utilizando bem o conceito de orientação a objetos, geram características notáveis para se utilizar o Spring na criação de Padrões Arquiteturais de Microsserviços.

Não satisfeito com tais vertentes, Spring agrega ainda mais valores os quais não foram citados, porém, podem ser vistos no livro "Vire o jogo com Spring Framework" de (WEISSMANN, 2014), que possui detalhes de como a ferramenta se comporta e quais problemas ela soluciona.

2.3.1.2 Docker (ANDERSON, 2015)

Outra ferramenta frequentemente utilizada é o Docker, uma plataforma que tem como finalidade facilitar a criação de aplicações, de forma que as mesmas funcionem como contêineres, hospedados em um Sistema Operacional, e que dividem recursos. Assim como citado por Anderson (2015) "Docker é uma tecnologia de virtualização de contêiner. É como uma máquina virtual muito leve. Além de criar contêineres, fornecemos

o que chamamos de fluxo de trabalho do desenvolvedor, que realmente ajuda pessoas a criar contêineres e aplicativos dentro de contêineres e depois compartilhá-los entre seus colegas de equipe."

Tal como dito por (ANDERSON, 2015) e pela descrição dada anteriormente, pode-se observar que, a atribuição dada pela ferramenta garante não só praticidade para a equipe de desenvolvimento e manutenção, como também agilidade no deploy e compartilhamento entre todos os membros das equipes. Esta também é uma ferramenta que visa isolar o software e prover a este todos os recursos necessários para que o mesmo funcione de forma independente do resto da aplicação, garantindo assim um pouco mais de segurança caso haja algum problema.

O Docker também possui um repositório chamado Docker Hub, onde é possível gerar versionamento das aplicações, fazendo assim com que, caso um deploy gere problemas nas aplicações, este seja rapidamente desfeito ao retornar à uma versão onde o erro não exista, fazendo assim com que as aplicações estejam sempre em funcionamento.

2.3.1.3 Computação em Nuvem (*Cloud Computing*)

Cloud Computing ou Computação na Nuvem pode ser considerada um modelo de negócios baseados em recursos computacionais, que possui como intuito mudar a forma de como se “adquire” TI. Ao invés de possuir servidores locais para responder a requisições, a Computação em Nuvem trabalha com grandes *Datacenters* que disponibilizam o acesso a estes servidores por meio de uma rede grande, como o caso da *World Wide Web*, também conhecida como internet. Temos alguns exemplos de computação em nuvem como como a gigante Amazon Web Services (AWS) (VERAS, 2013) e também a renomada Microsoft Azure (CHAPPELL *et al.*, 2009).

A AWS consiste em uma distribuidora de serviços que trabalham encima do conceito de Computação em Nuvem altamente escalável, a qual disponibiliza estes serviços na forma *on demand*, a qual você paga apenas pelo serviço que queira utilizar e são divididos em grupos de ferramentas previamente já definidas. Estas podem comunicar-se entre si de forma fácil, além de serem altamente escaláveis e fracamente acopladas o que gera uma melhor tolerância a falhas na aplicação.

Em outra grande empresa do mercado, a Microsoft Azure, vemos semelhanças na forma que a mesma foi criada, em relação a Amazon Web Services (AWS). Diversos serviços disponibilizados pela AWS, também podem ser encontrados na plataforma da Azure, porém com singularidades que não são retratadas em ambas as plataformas, a Azure possui um foco voltada ao armazenamento e redistribuição de conteúdo, além de avaliação do mesmo por meio de um mecanismo próprio em camadas.

Ambas possuem um foco em comum, porém cada uma possui sua visão de trabalho, mesmo que as duas trabalhem usando o serviço baseado em Computação na

Nuvem, explicada no tópico da AWS. Vale acrescentar que as duas não são as únicas a disponibilizar esses serviços em nuvem, mas sim as mais conhecidas e consagradas.

2.4 Análise das Arquiteturas

Conforme descrito anteriormente, ao desenvolver novos sistemas sem se pensar em melhorias futuras na Arquitetura Monolítica, grandes problemas podem surgir no acoplamento e manutenção destes sistemas, sistemas complexos demais geram um enorme retrabalho para o desenvolvedor, o que resulta em grande perda de tempo atualizando todo o sistema para inserir novas funcionalidades ou até mesmo pequenas alterações podem se tornar grandes problemas.

Porém, é válido ressaltar que, em sistemas que possuem uma complexidade baixa e caso não haja a intenção de melhoria ou acoplamento de outros módulos tendo a necessidade de gerenciar estas novas funcionalidades, a Arquitetura Monolítica se apresenta como uma excelente proposta, como em sistemas de chat, que possuem uma Arquitetura simples, sem muita necessidade de novas funcionalidades. Assim, não há necessidade de investir em uma Arquitetura como a de Microsserviços, onde o custo seria maior, mesmo que com um desempenho melhor, já que o sistema não receberá novas funções.

Na Arquitetura Orientada a Serviços (SOA) é introduzida a ideia de desenvolvimento, visando criar serviços ou alterando um sistema já existente dentro deste método. Assim, beneficia-se não só o desenvolvedor, mas também a qualquer indivíduo que venha a trabalhar com o sistema, seja durante o seu desenvolvimento ou já na produção, pois ao se separar o projeto em serviços definidos é facilitado a inserção de novas funcionalidades ou até mesmo a integração do sistema a outros, criando um único sistema composto por vários serviços que se complementam.

Porém, como dito anteriormente nos tópicos sobre Microsserviços e Arquitetura Orientada a Serviços, há aqueles que vêm ambas as topologias e dizem que são as mesmas, não só pela topologia, mas também em suas descrições. De acordo com Duarte (2017): "não seriam os Microsserviços a mesma coisa que SOA, mas algumas décadas atrás quando o padrão nasceu? Inclusive algumas empresas que dizem usar SOA programam-na da mesma forma que hoje chamamos de Microsserviços e isso não é ruim." e ainda, de acordo com o mesmo, "SOA significa muitas coisas diferentes e até contraditórias para diferentes pessoas. Via de regra, geralmente o que se vê é SOA sendo usado para integrar diferentes aplicações Monolíticas por meio de um ESB (*Enterprise Service Bus*). [...] não dá pra comparar isso com Microsserviços, principalmente considerando a complexidade do canal de comunicação".

Esta e outras visões sobre o tópico são abordadas por diversas pessoas com diferenças mínimas de descrição, e em todos os casos, acabam se divergindo que são Arquitetura diferentes e que deveriam possuir um mesmo propósito, mas atuam de maneiras completamente divergentes uma da outra, sendo assim, se tornando independentes em seus critérios de desenvolvimento, mesmo que uma possa ser utilizada juntamente a outra.

Fonte: (RICHARDSON, 2019).

	Microsserviço	SOA
Comunicação entre serviços	Comunicação direta entre os serviços utilizando protocolos como REST ou gRPC	Barramento de serviço único que utiliza protocolos como SOAP
Dados	Organização, Esquema e Dados únicos por serviço	Organização, Esquema e Dados compartilhados por toda a aplicação
Utilização	Serviços pequenos e simples	Grandes aplicações monolíticas

Tabela 2.1: Tabela comparativa entre Microsserviços e SOA

Esta Arquitetura possui fortes características da Arquitetura SOA, podendo ser vistas algumas semelhanças na Tabela 2.1 retirada do livro de (RICHARDSON, 2019), além de que, conforme citado por (O'GRADY, 2017): "Os Microsserviços são mais fáceis de desenvolver do que as alternativas Monolíticas, e vêm sem a bagagem dos padrões de SOA". Os Microsserviços retiram restrições que o SOA define como padrões, permitindo o desenvolvimento de estruturas de forma mais sucinta, sem restrições de módulos, facilitando o agregamento de novas funções e a manutenção no sistema sem prejudicar o usuário final, porém com o custo da necessidade de uma enorme experiência para desenvolver tais serviços.

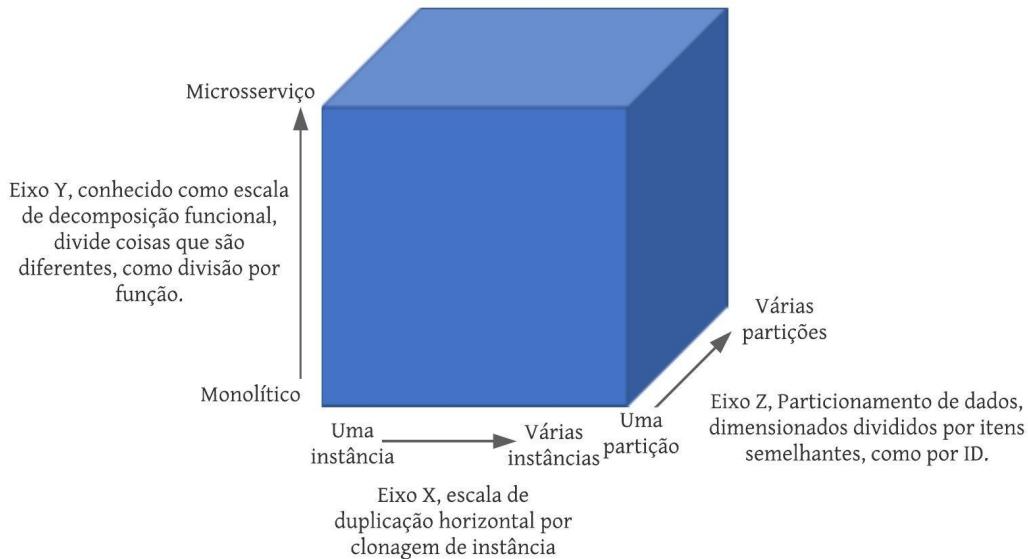


Figura 2.4: Diferentes dimensões para escalabilidade de uma aplicação.

Fonte: (RICHARDSON, 2019).

A Figura 2.4 retirada do livro de (RICHARDSON, 2019) é uma representação da escalabilidade de uma aplicação onde é definido de três maneiras diferentes para dimensionar um aplicativo, que são representados por eixos: o eixo X é o pedido de balanceamento de carga de escalonamento em várias instâncias idênticas; O eixo Z representa o escalonamento que roteia solicitações com base em um atributo da solicitação e o eixo Y decompõe funcionalmente um aplicativo em serviços.

Mas ao tratar de escalabilidade que seria um, se não o ponto mais forte, da Arquitetura de Microsserviços, temos que o padrão permite que requisitos não funcionais como escalabilidade e elasticidade, aumente e reduza instâncias de serviços de acordo com a necessidade, consumindo recursos físicos sob demanda e apenas para aquela funcionalidade que está sendo requisitada no momento. Tal funcionalidade não é possível de conseguir em Arquiteturas Monolíticas, que escalam tudo ou nada, consumindo de forma desnecessária recursos físicos e aumentando o valor da aplicação. Assim, também facilita a implementação de soluções complexas, possibilitando a divisão das aplicações para equipes diferentes para o seu desenvolvimento, diminuindo o custo e elevando o valor dos serviços, uma vez que a curva de aprendizado necessária para produzir as novas funcionalidades será bem menor.

2.5 Considerações Finais

Foi realizada uma discussão dos principais conceitos de algumas Arquiteturas de Software, pois o intuito deste trabalho é falar sobre os padrões de Microsserviços. Porém, a necessidade de mostrar as Arquiteturas sem dúvida era importante, demonstrando a quem esteja interessado o que são cada Arquitetura e suas interações entre as mesmas. Neste, é descrito especificamente sobre os Microsserviços, demonstrando as ferramentas utilizadas nessa Arquitetura.

Vale ressaltar que, ao revisar todas as ferramentas e padrões utilizados no desenvolvimento da aplicação, estas foram escolhidas com o intuito de diminuir o tempo de desenvolvimento, devido a menor necessidade de aprendizado para a criação das aplicações, uma vez que os padrões e frameworks utilizados já realizaram tais papéis. Assim, foi deixado apenas a ligação de tais padrões com as funções desejadas pelo usuário/cliente, diminuindo o custo das aplicações/serviços, dado o fato de ter sido utilizado funções em Nuvem, os quais dispensaram equipamentos físicos, substituídos pelas funcionalidades dos serviços da Amazon. Este serviço também pode permitir o uso de escalabilidade onde pode-se aumentar os recursos “físicos” do desenvolvimento para crescer a carga ou necessidade das aplicações ou reduzi-la para diminuir os gastos com o mesmo, caso não haja a necessidade dos equipamentos.

CAPÍTULO 3

Padrões para o Desenvolvimento de uma Arquitetura Voltada a Microsserviços

Este capítulo visa tratar sobre os padrões mais utilizados no desenvolvimento de Microsserviços, catalogar todos os que forem encontrados e em seguida utilizar alguns dos padrões no desenvolvimento de uma aplicação, utilizando-os.

Conforme Buschmann *et al.* (1996), as inspirações para se tratar padrões de software vieram principalmente do trabalho de Alexander (1977), que para este, um padrão consiste em "Uma regra de três partes, que se expressa em uma relação entre um determinado contexto, um problema e uma solução". Assim, pode-se entender que um padrão visa resolver um problema, levando em conta seu contexto.

No contexto de software podemos dizer que um padrão é composto por experiências anteriores no desenvolvimento de software. Essas experiências podem ser compostas por várias coisas como soluções para problemas cotidianos em softwares, técnicas de desenvolvimento e boas práticas de programação. Assim, os padrões consistem em uma espécie de manual de passos e técnicas a se seguir baseados nessas experiências similares, possibilitando que possa haver uma maior correlação para aqueles que estão desenvolvendo a aplicação, facilitando a manutenção da aplicação, realização de melhorias, correções de erros, e também a possibilidade de interligar novas funcionalidades mais facilmente.

Na Arquitetura de Microsserviços existem vários padrões, estes sendo enfoque deste trabalho. Como forma de buscar referências sobre padrões foi realizada uma pesquisa bibliográfica, tendo como ponto de partida o trabalho de Ghofrani e lübke (2018), sendo analisado os dados obtidos, referidos a padrões que eram aplicados em SOA, talvez aplicáveis, e os padrões aplicáveis a Microsserviços e SOA.

Originalmente, o trabalho continha os nomes dos padrões, porém não os descrevia e nem entrava em detalhes de como eram as suas aplicabilidades. Com isso, foi elaborado um catálogo composto por três partes, visando separar e definir estes padrões, organizados em: Aplicáveis em SOA / Não aplicáveis em MS, Talvez Aplicáveis a MS e Aplicáveis em SOA e MS.

O resultado é demonstrado no link, (NETO M. P. M; AUGUSTO, 2019). Em seguida, fora realizada uma nova revisão bibliográfica, a fim de encontrar mais padrões voltados especificamente a Microsserviços. Foram encontrados os trabalhos de (ARCITURA,), (RICHARDSON, a) e (TAIBI; LENARDUZZI; PAHL, 2018), que também tratavam sobre padrões.

Esse novo levantamento também foi incorporado no catálogo disponibilizado anteriormente, para tanto abaixo foi elencado os padrões que se repetiam em todas as referências encontradas. Sendo eles onze padrões, que em seguida foram distribuídos em três categorias, que se referem a que tipo de parte da aplicação o padrão pertence. Essas categorias e seus padrões consistem em:

- Integração:
 - Api Gateway.
 - Backends de API.
 - Token de Acesso.
- Banco de dados:
 - Banco de dados por serviço.
 - Tabelas privadas por serviço.
 - Esquema por serviços.
- Serviços:
 - Disjuntor.
 - Padrões de Descoberta.
 - Descoberta do lado do Cliente.
 - Descoberta do lado do Servidor.
 - Registro de serviço.

Essas categorias e padrões estão descritos abaixo.

3.1 Integração

Essa categoria consiste nos padrões que visam interligar os Microsserviços. Além disso, essa parte também é responsável por fornecer interfaces do sistema para o usuário.

3.1.1 *API Gateway*

A *API Gateway* consiste em um único ponto de entrada (Figura 3.1) para a aplicação, servindo como interface única de *API* para receber as requisições dos clientes

e encaminhar as mesmas para o seu serviço correspondente. As requisições podem ser agregadas aqui, a fim de fornecer um serviço único composto por vários Microsserviços.

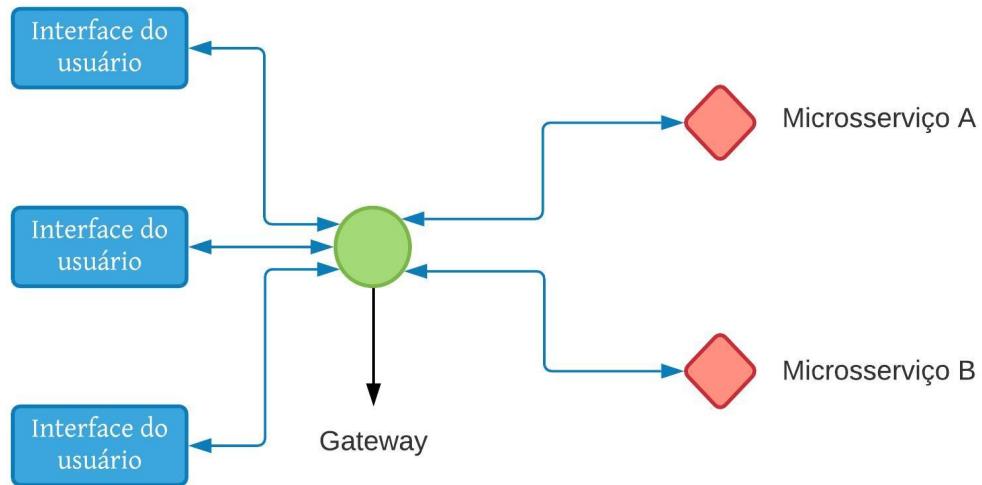


Figura 3.1: *Api Gateway fazendo o direcionamento das requisições para os microsserviços*

Fonte: Elaborada pelos autores.

3.1.2 *Backends de API*

Esse consiste em criar *Gateways* separados para cada tipo de cliente. Como exemplos, existem três tipos de clientes: aplicação web, aplicação móvel e aplicativo externo de terceiros, de forma que são criadas três *Gateways* de *API* diferentes, a fim de propor uma *API* para cada cliente específico. Esse padrão tem como finalidade disponibilizar os serviços de forma mais adequada a cada cliente.

3.1.3 *Token de Acesso (Access Token)*

A Figura 3.2 representa os passos de uma requisição feita por um usuário passando por um serviço de autenticação. O processo se inicia com o usuário fazendo uma requisição como login em uma aplicação, essa requisição demonstrada pelo item 1 envia a requisição para um ponto único de entrada, aqui anteriormente descrito como *API Gateway*. Por sua vez o *API Gateway* envia a requisição para o serviço de autenticação e de *token*, esse envio é demonstrado pelo item 2, o serviço de *token* recebe a requisição e a valida, tendo uma validação positiva a resposta é o *token* de acesso sendo este devolvido para o *API Gateway* pelo item 3. Após receber a validação positiva o *API Gateway* faz a requisição para o Microsserviço responsável pela requisição feita pelo usuário, sendo esta o item 4. Por fim, o Microsserviço executa a requisição e devolve a mesma para o *API Gateway* que por sua vez a devolve para o usuário, sendo esse o item 5.

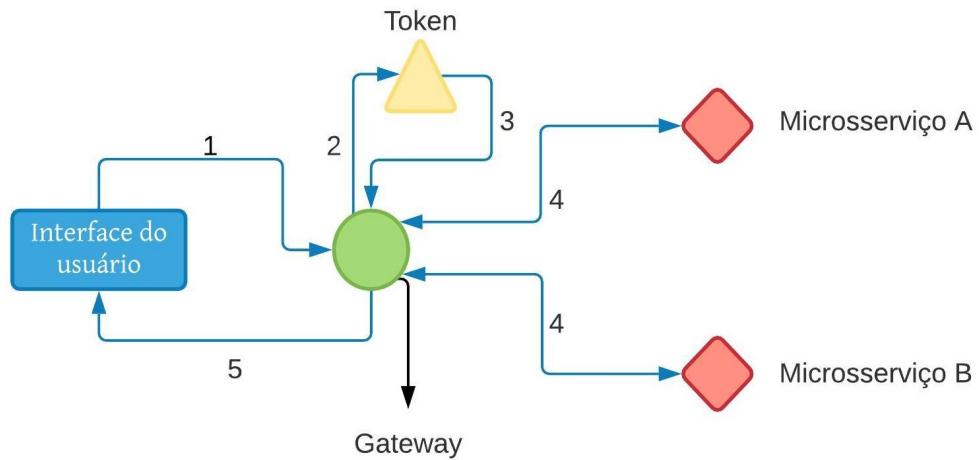


Figura 3.2: Controle de acesso e requisições utilizando um Token

Fonte: Elaborada pelos autores.

3.2 Banco de dados

Dentro do banco de dados existem especificamente três padrões, sendo eles: Banco de dados por serviço, Tabelas privadas por serviço e Esquema por serviço. Os mesmos são padrões que visam manter os dados persistentes de cada Microsserviços, privados para esse serviço, e acessíveis somente por meio de sua *API*. As transações de um serviço também devem envolver apenas o seu banco de dados. O banco de dados do serviço é efetivamente parte da implementação deste serviço fazendo parte de sua composição demonstrados na Figura 3.3.

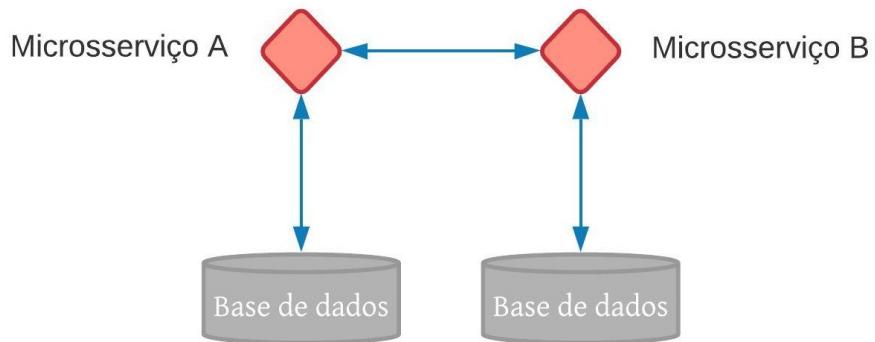


Figura 3.3: Banco de dados por serviço

Fonte: Elaborada pelos autores.

3.3 Serviços

Essa seção será tratada sobre os padrões que fazem referência a forma com a qual deve-se implementar os serviços, a fim de encontrar outros serviços disponíveis, bem como a possibilidade de gerar uma tolerância às falhas dentro da aplicação.

3.3.1 Disjuntor (*Circuit Breaker*)

Como podem existir muitas chamadas de serviço envolvidas em um aplicativo, uma falha em uma das chamadas pode causar uma cascata de erros. O padrão de disjuntor captura falhas e define um limite destas dentro de um tempo especificado, fechando o circuito caso esse limite exceda o aceitável. Tal ação, imediatamente relata um erro, e não permite mais que seja possível acessar o serviço. Existe também dentro do Disjuntor a possibilidade de adicionar o *Half-Open*, no qual o próprio serviço, após um tempo, valida se o problema ainda está ocorrendo, e caso seja detectada apenas uma falha, o mesmo volta a fechar o serviço. Do contrário o mesmo volta a ser acessível novamente.

3.3.2 Padrões de Descoberta (*Discovery Patterns*)

Em uma aplicação Monolítica, os serviços invocam um ao outro por meio de método próprios de suas linguagens ou chamadas de procedimento. Em uma implantação de sistema distribuído tradicional, os serviços são executados em locais fixos e conhecidos (hosts e portas) e, portanto, podem chamar cada um facilmente usando *HTTP/REST* ou algum mecanismo *Remote Procedure Call* (RPC). No entanto, um aplicativo moderno baseado em Microsserviços é comumente executado em ambientes virtualizados ou em contêiner, onde o número de instâncias de um serviço e suas localizações mudam dinamicamente. Consequentemente, os clientes de serviço devem estar habilitados para fazer solicitações para um conjunto de instâncias de serviços transitórios que mudem dinamicamente.

3.3.3 Descoberta do Lado do Cliente (*Client-Side Discover*)

Os clientes obtêm a localização de uma instância de serviço consultando um Registro de Serviços, que por sua vez, conhece os locais de todas as instâncias de serviço. Isso implica em menos partes móveis e saltos de rede, em comparação com a Descoberta do Lado do Servidor, porém os clientes são acoplados ao Registro de Serviço, necessitando assim, implementar uma lógica de descoberta de serviço do cliente para cada linguagem ou estrutura de programação usada pelo aplicativo.

3.3.4 Descoberta do Lado do Servidor (*Server-Side Discover*)

Ao fazer uma solicitação a um serviço, o cliente faz uma requisição por meio de um roteador (também conhecido como balanceador de carga). O roteador consulta um registro de serviço, que pode ser incorporado ao roteador, e encaminha a solicitação à uma instância de serviço disponível.

3.3.5 Registro de Serviço (*Service Registry*)

Um registro de serviço é um banco de dados de serviços, suas instâncias e seus locais. As instâncias de serviço são registradas no Registro de Serviço na inicialização, e canceladas no encerramento. O cliente do serviço ou os roteadores consultam o registro de serviço para localizar as instâncias disponíveis de um serviço.

3.4 Considerações Finais

Neste capítulo foram tratados alguns dos padrões utilizados para o desenvolvimento de uma Arquitetura Orientada a Microsserviços. Os padrões citados tratam-se dos mais comumente utilizados na Arquitetura e que geralmente servem de base para o desenvolvimento de Microsserviços.

Vale ressaltar que, conforme o catálogo demonstrado, existem outros padrões de Microsserviços. É importante reafirmar que é necessário realizar uma análise visando encontrar os padrões que são cabíveis ou não para a aplicação com a qual se deseja desenvolver na Arquitetura Orientada a Microsserviços. Há também aquelas aplicações das quais se utilizam destes padrões apenas como base, tendo em vista que os padrões descritos anteriormente são bem escaláveis trazendo bastante vantagens em aplicações.

CAPÍTULO 4

Estudo de caso sobre a aplicação de padrões de microsserviços

Este capítulo tem como objetivo demonstrar a aplicabilidade prática na aplicação de alguns dos padrões descritos no Capítulo 3, além de explorar e demonstrar as vantagens e desvantagens do uso dos padrões. Tendo isso como premissa, iniciou-se a busca de uma aplicação Monolítica que pudesse ser convertida para Microsserviços e receber tais padrões, mantendo seu propósito e evitando o tendencionismo para uma ou outra Arquitetura em sua construção.

Sendo assim, foi realizada uma busca em repositórios e em sites, por aplicações ou pedaços de códigos, que pudessem ser alterados da Arquitetura Monolítica para a Arquitetura de Microsserviços ou vice e versa, ou então adaptadas a receber os padrões descritos anteriormente.

Depois de serem considerados vários projetos, foi escolhido um projeto proposto pela Amazon Dividir... (2018), do qual consistia em criar uma aplicação Monolítica e em seguida converter a mesma para Microsserviços e por fim, receber os padrões propostos anteriormente. Do projeto original proposto pela Amazon foi utilizado apenas as partes que faziam a criação do serviço na nuvem, da configuração das máquinas e a organização das pastas do projeto. O restante do código em sua parte funcional foi totalmente reescrito e adaptado, principalmente no caso da Arquitetura de Microsserviços, onde a mesma recebeu alguns dos seus padrões como o de Disjuntor e de Banco de dados por serviço.

A escolha levou em consideração alguns fatores como a possibilidade do código proposto pela Amazon ser disposto a rodar todo o projeto em uma infraestrutura de nuvem, além de alguns problemas com ambientes locais (capacidade computacional, problemas com rede, etc) não interferirem na aplicação. Vale ressaltar que ambos os projetos iriam utilizar dos mesmos recursos disponibilizados pelo AWS, novamente uma forma de evitar tendencionismos para alguma Arquitetura, e também o fato de que a aplicação proposta pela Amazon utilizaria duas ferramentas (*Docker* e Computação em Nuvem) discutidas na secção 2.3.1 - **Ferramentas Utilizadas no Desenvolvimento de Microsserviços**.

4.1 Caracterização da aplicação

A construção de ambas aplicações, seguiu um modelo descrito pela AWS Dividir... (2018), em que havia um tutorial propondo a quebra de um serviço Monolítico para um com Microsserviços. A aplicação em si consistia em um sistema de compras, em que haviam as classes de Clientes, Mensagens de Compras, na qual todos os usuários poderiam ser clientes e vendedores ao mesmo tempo. Todos eles podiam realizar algumas ações demonstradas pelo diagrama de caso de uso na Figura 4.1.

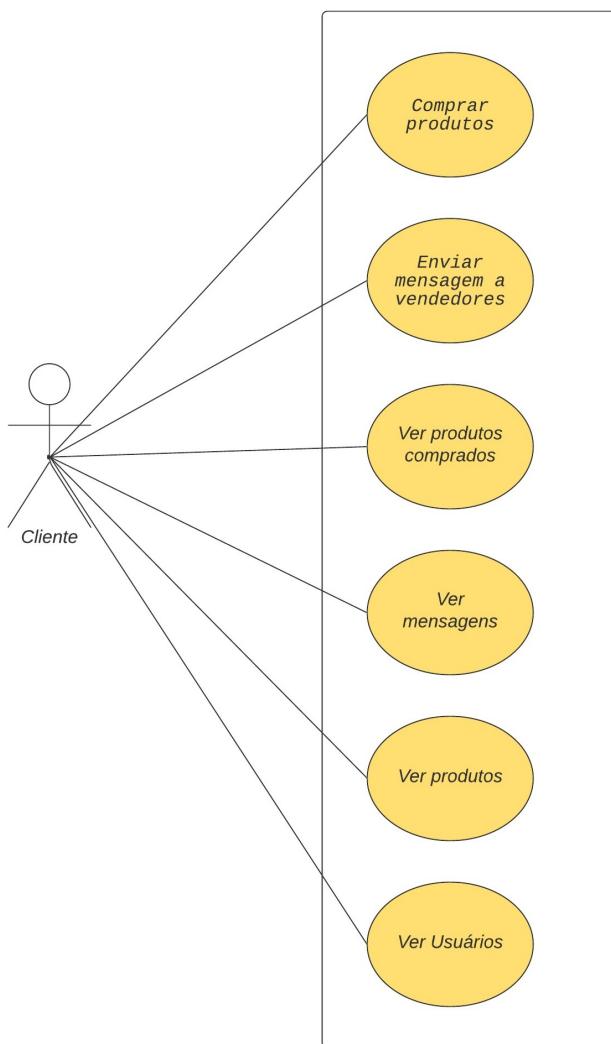


Figura 4.1: Diagrama de caso de uso

Fonte: Elaborada pelos autores.

Já a parte de código em si estava escrito em *JavaScript* em conjunto com o interpretador *Node.js*. O código encontrado foi reescrito pelos autores, visando adicionar chamadas a bancos de dados que foram criados na nuvem e a implementação do padrão de Disjuntor no caso do código da aplicação que utilizava os Microsserviços. Porém,

vale ressaltar que, nenhum dos serviços possuía uma real interação entre eles, ou seja, nenhuma função chamava outra dentro dos sistemas.

4.2 Metodologia de implantação

O processo de implementação da aplicação na nuvem iniciou-se na criação de uma conta na AWS para poder realizar a construção da aplicação. Após a ativação da conta, foi realizada a construção de ambas as Arquiteturas seguindo o modelo proposto da Amazon, mas utilizando a aplicação reescrita. Ambas as Arquiteturas compartilhavam de característica infraestruturais em comum, sendo que ambas rodavam em um Cluster (hospedado na região US East - Ohio) composto por 2 máquinas virtuais iguais (t2.micro) e utilizando um único balanceador de carga. As máquinas não tiveram nenhuma configuração de escalabilidade para os testes, tendo essa decisão sido tomada visando manter os custos da aplicação dentro do limite gratuito de uso disponibilizado pela Amazon.

O código do serviço Monolítico era composto em um único contêiner do *Docker*, já o código de Microsserviços era dividido em três serviços. Sendo assim, foi necessário que na configuração dos serviços dentro do *Cluster* fosse definido um serviço para o Monolítico e três serviços (um para cada contêiner) dos Microsserviços, onde a ativação ou inativação desses serviços definiria qual Arquitetura seria executada dentro do *Cluster*.

Juntamente com essa definição, também foi necessário criar definições de tarefas (uma para cada contêiner), que consistiam em definir parâmetros e os contêineres a serem usados pelos serviços. Sendo definido que cada contêiner iria ter o máximo de 256 mb de memória, a porta 3000 (por padrão do servidor *Node.js*) que seria buscada no contêiner e a quantidade de créditos de processamento definidos como 256 vCPU que consistem em cada crédito (vCPU) na utilização de 100 porcento de um núcleo do processador durante um minuto.

Essa caracterização é uma visão geral da aplicação que foi construída dentro da Nuvem do AWS. A utilização da nuvem possibilitou com que os desenvolvedores focassem mais na aplicação, sendo que a parte da nuvem em si foi realizada de forma extremamente simples, e como resultado havia a disposição toda a infraestrutura necessária para o bom funcionamento da aplicação sem terem que se preocupar com a sua infraestrutura e escalabilidade. No próximo parágrafo será detalhada como foi a utilização dos serviços que foram utilizados na construção das Arquiteturas.

4.3 Serviços utilizados para compor a aplicação

Como vantagem de se utilizar o Amazon Web Services, temos que o mesmo já disponibilizava vários serviços que possibilitaram a construção, com fácil integração

entre si, bem escaláveis e que o desenvolvedor não precisasse ficar se preocupando em integrar partes de código para que os mesmos se comuniquem entre si. Ambas aplicações foram compostas utilizando alguns serviços da Amazon em comum, mas tendo alguns diferentes, levando em consideração se o mesmo era aplicável ao padrão Arquitetural de cada aplicação.

A composição desses serviços foi subdividida, a fim de separar os serviços que foram utilizados visando demonstrar algum padrão de Microsserviços proposto no Capítulo 3 e os que foram utilizados com outra finalidade como monitoramento e aplicação de políticas nos serviços.

4.3.1 Serviços do AWS utilizados na construção das arquiteturas

Amazon API Gateway: Consiste em um serviço que disponibiliza ao desenvolvedor, várias possibilidades de controle e gerenciamento de APIs. O mesmo trabalha como sendo porta de entrada para requisições que querem acessar serviços dentro da nuvem. O mesmo também consegue gerir o tráfego de requisições e aplicar políticas de controle de acesso e autorização. Esse serviço da Amazon possibilitou a aplicação de dois padrões propostos, o padrão de uma API Gateway servindo como porta de entrada única para aplicação e também o padrão de Token de acesso para controlar o acesso às requisições que iriam ser feitas a aplicação. O mesmo não é utilizado na aplicação Monolítica.

Amazon Relational Database Service (RDS): Serviço que consiste em um banco de dados distribuído. No caso da aplicação foi utilizado instâncias rodando o sistema de gerenciamento de bancos de dados *MySQL*. Esse serviço foi utilizado para criar bancos de dados para cada Microsserviço criado, conforme o padrão de banco de dados. Vale ressaltar que esse serviço também foi utilizado na Arquitetura Monolítica.

Amazon Elastic Container Service (ECS): O serviço ECS tem por sua finalidade a orquestração e o controle de contêineres *Docker*, possibilitando criar e executar aplicações conteinerizadas dentro do AWS. O mesmo tem integração com vários outros serviços, sendo possível visualizar isso nos serviços descritos abaixo. O ECS foi utilizado aqui para além de orquestrar os contêineres servir como base central da aplicação, e usado também para aplicar os padrões de Registro de serviço, descoberta do lado do servidor e Padrões de descoberta, trabalhando em conjunto com o Amazon Route53 para executar tais padrões. Dentro do ECS também foi realizada todas as configurações necessárias para integrar toda a infraestrutura criada. No mesmo foi criada as *Task Definition*, que definem configurações que o contêiner será utilizado e os detalhes do mesmo, como os recursos que o mesmo irá ter é definido os *Services* que é de fato a aplicação rodando. Essa parte também é onde se define as interligações da aplicação como o balanceamento de carga e a descoberta e registro de serviço. Os *Services* trabalhando juntamente com

a *Task Definition* recebendo suas informações e integrando na infraestrutura, a fim de realizar a orquestração da execução da aplicação.

Amazon Route53: Este serviço é um serviço de DNS, que serve como local para receber solicitações que desejam acessar as aplicações dentro da nuvem. O mesmo foi utilizado para trabalhar em conjunto com ECS, aplicando os padrões de Registro de serviço, descoberta de serviços pelo lado do servidor e Padrões de descoberta. Para realizar isso o ECS comunica com Route53, a fim de acessar o DNS criado, sendo este a porta de acesso e descoberta para os serviços a serem acessados. Essa comunicação é escalável, possibilitando aumentar a capacidade no que desrespeita a achar e criar mais instâncias interligadas ou reduzi-lás. O mesmo não é utilizado na aplicação Monolítica.

Amazon Elastic Container Registry (ECR): Este é um serviço para o armazenamento e gerenciamento de imagens de contêineres *Docker* que são utilizadas pelo ECS. Esse serviço serviu para armazenar a aplicação e criar contêineres para cada Microsserviço ou um contêiner único para o caso do Monolítico.

Amazon CloudFormation: Este serviço serve como criação de um conjuntos de recursos de infraestrutura que são rapidamente criados e ficam disponíveis para executar aplicações. É uma forma onde o desenvolvedor passa alguns parâmetros e recursos que ele deseja utilizar, e a nuvem prepara toda a infraestrutura para o desenvolvedor, bastando o mesmo apenas focar em sua aplicação.

Amazon Elastic Compute Cloud (EC2): Consiste no serviço que disponibiliza capacidade computacional dentro da nuvem, sendo essas máquinas altamente escaláveis e com várias configurações disponíveis, a fim de atender especificamente cada demanda. O EC2 foi utilizado para rodar a aplicação mantida pelo ECS, de forma que o mesmo a integrava na criação do *Services*. No caso desse modelo, foram utilizadas duas instâncias t2.micro (Sistema operacional Linux, 1GB de Memória Ram e CPU de 1 Núcleo) trabalhando juntas em um *Cluster*.

Amazon Elastic Load Balancing (ELB): Serviço que atua como balanceamento de carga, sendo esse responsável por receber e distribuir as requisições para seus respectivos *endpoints*. Esse serviço serviu como forma de organizar o acesso dos *Services* ao processamento das instâncias EC2 e também como ponto para retorno das respostas dos processamentos advindos das máquinas. No caso da aplicação Monolítica, o mesmo serviu como porta de entrada para as requisições.

4.4 Arquitetura da Aplicação monolítica

Na construção da Arquitetura Monolítica foram utilizados os serviços de ELB, ECS, EC2, ECR e RDS. Também foi definida a criação de uma *Task Definition* e

um *Services*. Similarmente, foi incluído o serviço CloudWatch. O resultado final da Arquitetura Monolítica pode ser visto na Figura 4.2:

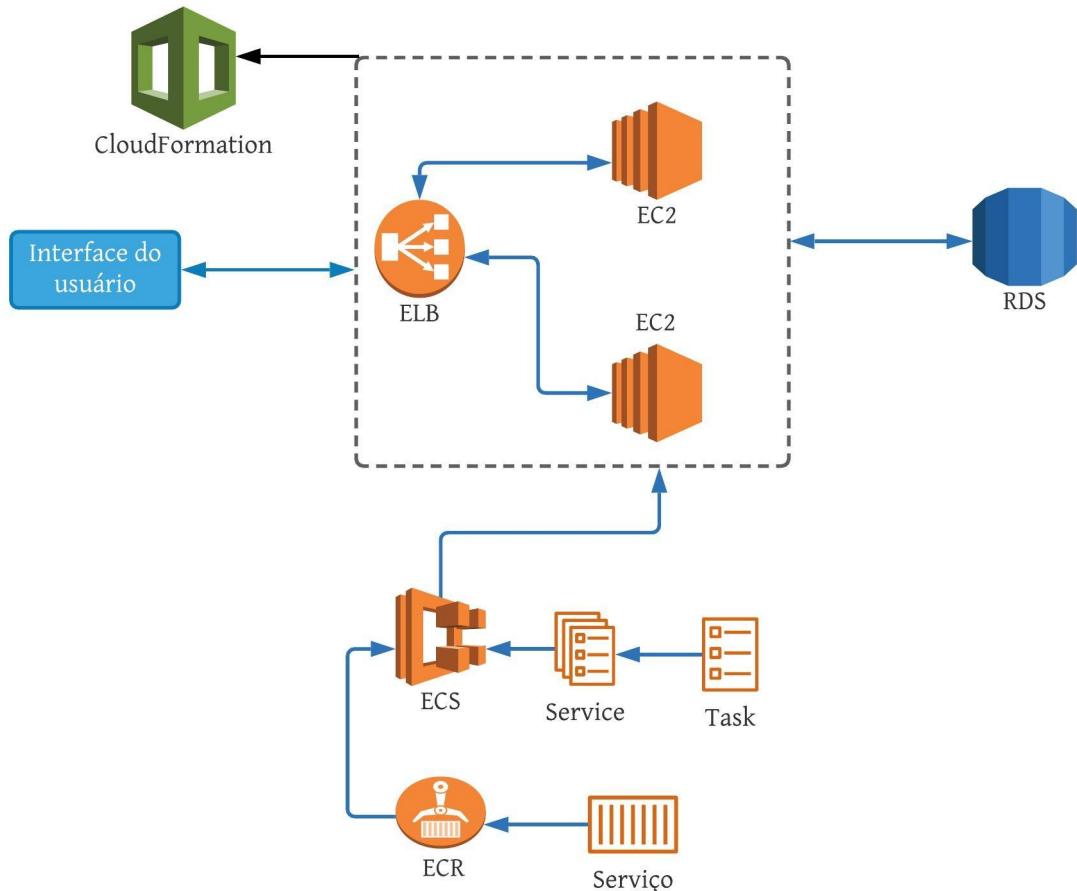


Figura 4.2: Arquitetura Monolítica implantada na AWS

Fonte: Elaborada pelos autores.

O fluxo da aplicação Monolítica consiste em, o usuário fazer uma requisição que é enviada para o Elastic Load Balancing, e esse por sua vez balanceia as entradas buscando a máquina que esteja mais disponível, para assim em seguida executar a chamada do *Services* ECS, que em seguida é executado em uma instância EC2, e após a execução é devolvido o resultado para o solicitante.

4.5 Transformação da Aplicação Monolítica para Microsserviços com seus padrões

Visando realizar esta transformação foi realizada a quebra do Monólito em três Microsserviços, e em seguida foram feitas as configurações necessárias para que cada

serviço se comportasse como único. Dessa forma, a parte estrutural interna da aplicação é basicamente a mesma da Monolítica, mas com três serviços com características únicas a eles. Em seguida foi feito a adição dos padrões de Microsserviços encontrados em forma de serviços oferecidos pela Amazon.

Como serviços utilizados temos o serviço de *API Gateway*, servindo este para aplicar o padrão de nome correlato. Vale ressaltar aqui que o mesmo serviço também poderia ser utilizado para o padrão de Backend de API, entretanto no caso específico do projeto, o mesmo não foi implementado, tendo em vista que ele não era aplicável ao escopo definido anteriormente.

O serviço Route53 foi utilizado para trabalhar em conjunto com ECS, aplicando os padrões de Registro de serviço, descoberta de serviços pelo lado do servidor e Padrões de descoberta.

O padrão de banco de dados único (Esquemas, Bancos e Tabelas) por Microsserviços foi aplicado utilizando o serviço RDS da Amazon.

No código da aplicação foi utilizado um *middleware*, para realizar o papel de Disjuntor, juntamente com um cachê em JSON, que armazena a última consulta bem sucedida.

Também foram utilizados alguns serviços comuns à aplicação Monolítica na transformação para a Arquitetura de Microsserviços, como os serviços de ELB, EC2 e *CloudFormation*. Para a execução dos serviços na nuvem foi definida a criação de uma *Task Definition* e um *Services*, sendo que foi feito uma de cada para cada Microsserviço. E para cada Microsserviço foi criado um contêiner *Docker*, sendo eles armazenados no ECR, serviço para o qual tem esse fim.

O resultado final da Arquitetura de Microsserviços com seus padrões pode ser visto na Figura 4.3:

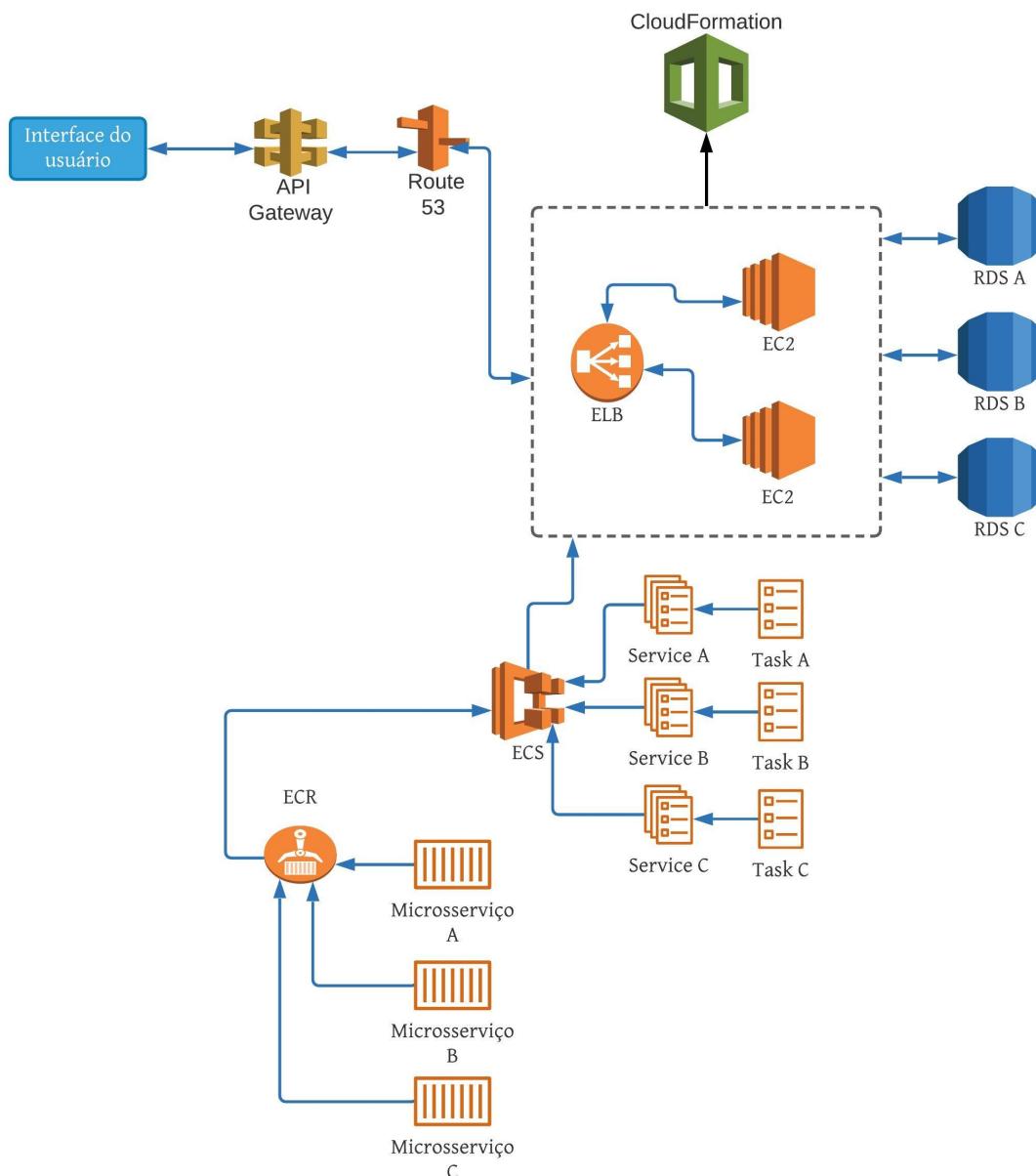


Figura 4.3: Arquitetura de Microsserviços implantada na AWS

Fonte: Elaborada pelos autores.

O fluxo da aplicação de Microsserviços consiste em: a requisição é feita pelo usuário que em seguida é enviada para o API Gateway, que por sua vez chama uma URL definida pelo Route53 como uma porta de entrada para o Elactic Loab Balancing. Este, por sua vez, balanceia as entradas buscando a máquina que esteja mais disponível, para em seguida o chamar o *Services* do ECS referente àquela URL buscando seu DNS, para em seguida achar a *Task Definition* específica àquela requisição, para enfim ser executado em uma instância EC2, e após a execução é devolvido o resultado para o solicitante.

4.6 Considerações Finais

Nesta seção foi descrita a criação de duas aplicações idênticas no seu propósito, mas com construções Arquiteturais diferentes. Agora, após a construção de cada Arquitetura, no capítulo seguinte, as mesmas serão submetidas a avaliações, que visam obter dados de ambas Arquiteturas, tendo como propósito final mostrar que a Arquitetura de Microsserviços com padrões se demonstra mais vantajosa que a Arquitetura Monolítica.

CAPÍTULO 5

Avaliação

Este capítulo tem como objetivo avaliar comparativamente as Arquiteturas Monolítica e de Microsserviços propostas no Capítulo

Para tal, foi feito um estudo buscando selecionar alguns testes que poderiam mostrar a forma de como cada Arquitetura se comportaria em um ambiente real. Nesses estudos conduzidos foi encontrado o trabalho de Akbulut e Perros (2019) onde os mesmos também utilizaram de uma implementação em *Node.js*, e em seus trabalhos realizaram alguns testes e considerações feitas sobre a análise dos resultados.

Dessa forma, conduziu-se a avaliação e testes baseados na forma como o estudo foi feito, além de ser acrescentado na avaliação a questão do custo e da tolerância a falha de cada implementação.

Assim, foi feito um escopo de testes visando explorar os aspectos de quantas requisições cada aplicação conseguiria atender em um determinado período de tempo, tempo de resposta das requisições, comparação entre os custos de cada aplicação considerando um serviço de nuvem, comparação entre o tempo necessário para desenvolvimento de cada Arquitetura e por fim, a tolerância às falhas de cada uma.

Visando com que os testes que se referiam ao acesso à aplicação, como o de quantidade total de requisições e de tempo de resposta, fossem mais realísticos, foi realizado um levantamento buscando informações sobre como é o tráfego de acesso em determinado site em um período de tempo. Nesse estudo foi encontrado o trabalho de Urdaneta, Pierre e Steen (2009), onde os pesquisadores fizeram um estudo rastreando o tráfego de acesso na Wikipédia durante 5 meses (entre Setembro de 2007 a Janeiro de 2008).

Desse estudo, foi feito uma análise dos dados da pesquisa levando em conta apenas os dados que eram referentes das 08:00 às 18:00 horas e apenas em dias úteis. Assim foi selecionado o mês de Outubro para servir como base, por ser um mês que havia mais dados disponíveis. Nesse mês foi observado que no período de tempo proposto havia em média de 100 a 140 mil requisições feitas durante o período de 1 minuto. Desta forma, foi considerada que a aplicação deveria ter a capacidade de atender algo entorno a 100 a 140 mil requisições para obter os dados demonstrados dos testes de forma mais realística.

Esse capítulo reúne o que foi descrito anteriormente, como preceitos para seu desenvolvimento. Após tais preceitos, foi levantado a necessidade de uma ferramenta para executar boa partes dos testes propostos, sendo este levantamento descrito abaixo.

5.1 Avaliação da ferramenta dos testes

Os testes foram decididos pelos membros do projeto, sendo acertado as ferramentas e tipos específicos de testes que seriam abordados durante a fase de avaliação. Em relação à ferramenta utilizada durante os testes, foi realizado um estudo sobre todas as possibilidades viáveis como a *LoadComplete*, *Postman*, *Tsung* e por fim, a ferramenta escolhida para a avaliação das Arquiteturas foi o *BlazeMeter*, que é a ferramenta *Apache JMeter* porém em sua versão na nuvem. A escolha foi realizada pois a grande parte das ferramentas não se enquadram nos testes pretendia-se se realizar, o qual o *BlazeMeter* se adequou com sucesso. Em relação ao não usar o *JMeter*, teve-se como fato da interface e da praticidade que sua versão da nuvem proporciona para o usuário que está realizando os testes.

5.2 Seleção dos testes realizados

Em relação aos testes decididos para a avaliação da estrutura das Arquiteturas, foram decididos os seguintes testes: Teste de carga, Teste de performance e Tolerância a falhas. A escolha dos testes realizados teve como objetivo avaliar aspectos essenciais sobre aplicações distribuídas, considerando um cenário real.

5.2.1 Teste de Carga

O teste de carga consistiu em uma bateria de testes realizados em certos períodos de tempos que foram determinados por um pequeno estudo, onde foi possível visualizar uma diferença válida entre um teste e outro em relação as requisições feitas aos serviços web. Tais testes consistem em uma variação de requisições HTTP feitas aos servidores na nuvem aos quais hospedam os serviços. O teste foi definido da seguinte maneira:

- 50 Usuários (VMs criadas pela ferramenta).
- Limitação de 1000-3000 requisições por segundo para cada usuário.
- Requisições feitas pelo servidor localizado em US East (Ohio) - Amazon Web Services.
- A duração dos testes consistia no intervalo de tempo de 2 minutos em cada um dos testes com variação na quantidade de requisição limitada por teste.

Estas restrições/condições do teste se deram necessárias por alguns fatores: verificação da disponibilidade do serviço, que seria o meio em que os serviços iriam atuar e mostrar caso eles tivessem alguma variação de resultados exponencial quando aumentado o limite de requisições, além do tempo limitante. Decidiu-se colocar o teste sobre um tempo limite para manter a comparação viável para que não haja discrepância no resultado caso o tempo seja diferente entre os testes.

Assim se prossegue para os resultados encontrados, que podem ser vistos na Figura 5.1.

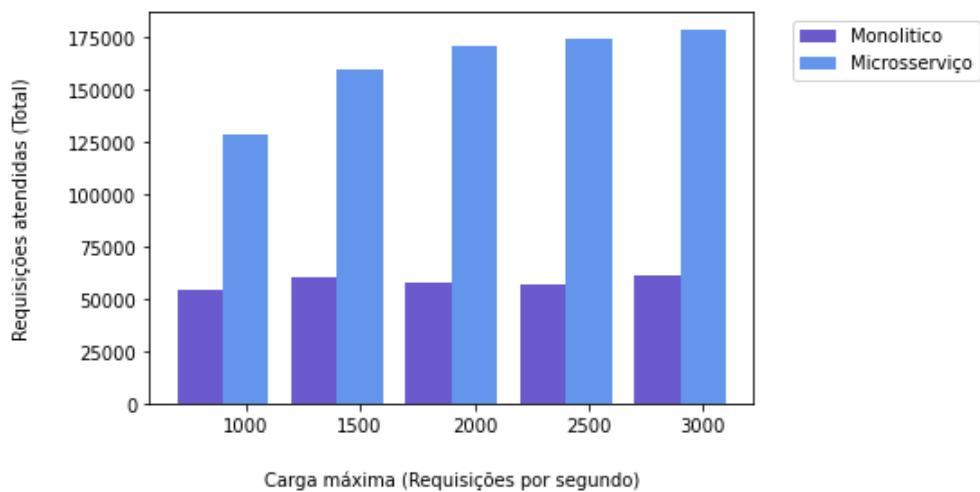


Figura 5.1: Teste de carga

Fonte: Elaborada pelos autores.

Onde pode-se observar que, em todos os cenários (1000-3000 requisições) a quantidade de requisições feitas e aceitas pela Arquitetura de Microsserviços foi superior em relação a Arquitetura Monolítica, em que cada um dos cenários a discrepância só aumentava em relação a quantidade de requisições feitas para ambas. Sendo assim, podemos deduzir que, ao final de todas as interações de requisições feitas para ambas as Arquiteturas durante os testes, os serviços baseados em Microsserviços se colocaram em vantagem em relação a requisição e resposta em cima dos serviços Monolíticos, mostrando assim que os padrões implementados permitiram que os serviços atendessem às requisições de forma independente sem ter que chamar outros serviços, os quais não possuíam necessidade direta pedida pelo usuário.

5.2.2 Teste de performance

O teste de performance consistiu na capacidade de ambas Arquiteturas responderem às requisições e o tempo hábil que levaram para responder todas elas durante o teste de carga citado anteriormente. Assim podemos julgar qual delas possui uma aceitação ou dinâmica melhor para receber requisições direcionadas a elas e ser capaz de aceitar e

entregar a resposta (no caso o acesso as páginas dos serviços), de forma que os usuários possam acessar o mais rápido possível os serviços.

Este teste foi realizado ao observar a quantidade de requisições que eram atendidas por segundo em cada uma das interações seguindo as restrições:

- 50 Usuários (VMs criadas pela ferramenta).
- Limitação de 1000-3000 requisições por segundo para cada usuário.
- Requisições feitas pelo servidor localizado em US East (Ohio) - Amazon Web Services.
- A duração dos testes consistia no intervalo de tempo de 2 minutos em cada um dos testes com variação na quantidade de requisição limitada por teste.

Assim podemos observar na Figura 5.2.

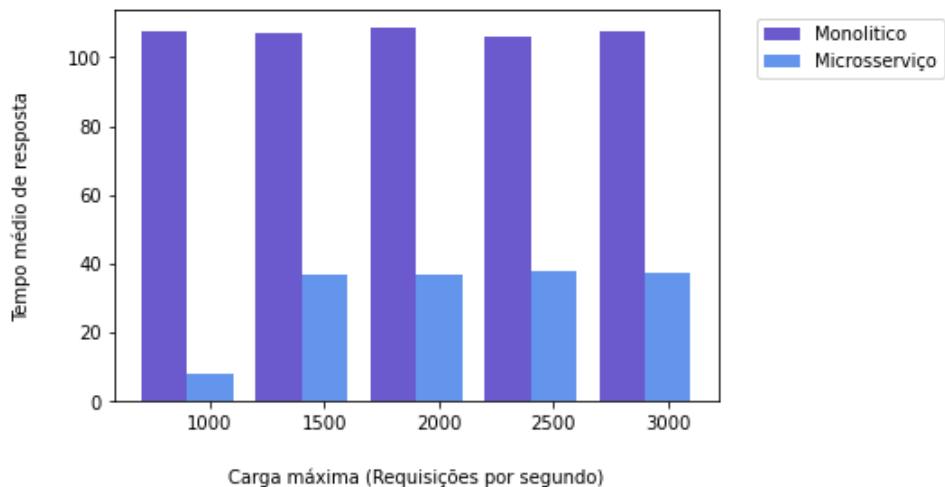


Figura 5.2: Teste de performance

Fonte: Elaborada pelos autores.

Na Figura 5.2 pode-se observar que, em cada rodada de teste houve um aumento no tempo de resposta das requisições por parte do padrão Arquitetural de Microsserviços, sendo que o padrão Arquitetural Monolítico seguiu uma certa estabilidade durante os testes, onde possui uma média durante todos os testes.

Essa estabilidade se deve ao fato de que a própria nuvem da AWS, começou a não permitir o recebimento de novas requisições enquanto as que estavam sendo processadas fossem resolvidas. Esse bloqueio é um sistema que a própria nuvem implementa para evitar problemas de ataques de negação de serviço, visando exaurir a aplicação, e consequentemente, tal comportamento limitou que o serviço não recebesse mais requisições, causando um tempo de resposta igual em todos os volumes de carga enviados.

É possível observar quer ambos os padrões passaram pela mesma carga de testes, porém o tempo de resposta do padrão Arquitetural Monolítico é muito maior que o padrão

Arquitetural de Microsserviços, mostrando assim uma vantagem visível ao utilizar tais padrões em questão ao tempo de resposta de requisições.

5.3 Avaliação sobre tolerância a falhas

Conforme Weber (2002) "O objetivo de tolerância a falhas é alcançar dependabilidade. O termo dependabilidade é uma tradução literal do termo inglês dependability, que indica a qualidade do serviço fornecido por um dado sistema e a confiança depositada no serviço fornecido".

A autora Weber (2002) complementa também que os "Principais atributos de dependabilidade são confiabilidade, disponibilidade de segurança de funcionamento (safety), segurança (security), manutenibilidade, testabilidade e comprometimento do desempenho (performability)".

Esses atributos são bastante relevantes quando se trata a questão de um sistema tolerante a falhas. Assim, das Arquiteturas propostas nesse experimento, podemos ver que a Arquitetura com padrões de Microsserviços, consegue se destacar por ter mais pontos em todos os atributos descritos anteriormente.

Desses pontos podemos citar a disponibilidade, principalmente pelo fato de que mesmo que um serviço deixe de funcionar, os outros continuam funcionando normalmente, a manutenibilidade, pelo fato de que caso seja necessário realizar algum tipo de manutenção a mesma pode ser realizada apenas em um pedaço do código sem ter que ficar recriando todo o projeto novamente e a performance que conforme os testes anteriores, a Arquitetura de Microsserviços se mostrou com um desempenho bem melhor que comparada a Monolítica.

Ressalta-se que não foram realizados testes visando avaliar a tolerância a falhas de cada Arquitetura, devido que os testes poderiam ficar muito tendenciosos, principalmente pelos pontos descritos anteriormente. Outros testes que talvez pudessem ser executados, levariam muito trabalho para serem construídos e teriam um custo bastante elevado.

5.4 Avaliação de custo das arquiteturas

Como mencionado anteriormente o objetivo ao construir as Arquiteturas Monolíticas e de Microsserviços sempre foi de se manter no que era disponibilizado gratuitamente pela AWS.

Abaixo é descrito um cenário onde imagina-se que o serviço tenha a demanda de aproximadamente 30 mil requisições por minuto na Arquitetura Monolítica e 90 mil requisições por minuto na Arquitetura de Microsserviços, tendo essa demanda 24

horas por dia, durante 30 dias (1 Mês). As demandas de requisições levam em conta a quantidade máxima de requisições que cada Arquitetura atendeu no período de 1 minuto no teste de carga descrito anteriormente.

Todos os custos foram calculados usando a ferramenta *AWS Pricing Calculator*, sendo definida como região padrão para os serviços que haviam essa necessidade a US East (Ohio). Os resultados dos custos estão descritos nas tabelas 5.1 e 5.2 a seguir.

Fonte: Elaborada pelos autores.

Monolítico			
Serviço	Parametro	Quantidade	Valor
Api Gateway	Não aplicável	Não aplicável	Não aplicável
ECR	1 Container referente ao tamanho da aplicação disponível por 1 mês.	45mb	USD 0.0044
EC2	Instancias t2.micro Linux, disponíveis por 1 mês.	2	USD 4.964
RDS for MySQL	Instancia t3.Micro com 1GB de armazenamento, reservada por 1 ano, mas tendo seu custo apurado mensalmente.	1	USD 4.28
ELB	1 Application Load Balancer, processando GB totais durante 1 mês. Levando em conta o tamanho médio de cada resposta como 100kb.	132.192	USD 1,073.96
Total: USD 1.083,20			

Tabela 5.1: Tabela de custos Monolítico

Fonte: Elaborada pelos autores.

Microsserviço			
Serviço	Parametro	Quantidade	Valor
Api Gateway	Quantidade total de requisições feitas por minuto em 1 mês (30 dias) e 1032 Kb para tamanho máximo de cada requisição.	3.855.081.600	USD 10,438.72
ECR	3 Container referentes a cada microsserviço, disponível por 1 mês.	100mb	USD 0.0098
EC2	Instancias t2.micro Linux, disponíveis por 1 mês.	2	USD 4.964
RDS for MySQL	Instancia t3.micro com 1GB de armazenamento, reservada por 1 ano, mas tendo seu custo apurado mensalmente.	3	USD12.83
ELB	1 Application Load Balancer, processando GB por hora durante 1 mês. Levando em conta o tamanho médio de cada resposta como 100kb.	385.344	USD 3,099.18
Total: USD 13.555,70			

Tabela 5.2: Tabela de custos Microsserviço

Ao observarmos as tabelas podemos ver que a Arquitetura de Microsserviços tem um custo bem maior, aproximadamente 13 vezes mais cara. Esse custo mais elevado é causado principalmente pelos serviços de API Gateway e ELB, que em comum, são serviços cobrados sobre requisições. Como a Arquitetura de Microsserviços atende mais requisições que a Arquitetura Monolítica, é esperado que esse valor de fato seja maior nessa infraestrutura.

A avaliação aqui realizada visou mensurar qual seria o custo máximo de cada Arquitetura, utilizando a infraestrutura da AWS, sendo usado como base para a mensuração dos valores e a quantidade máxima de requisições atendidas por cada uma no teste de carga. Fato este que geralmente não é comum, podendo os custos mensurados aqui em um ambiente de produção ser menores ou maiores caso haja uma variação na demanda pela aplicação.

5.5 Considerações Finais

Neste capítulo realizada a avaliação das duas Arquiteturas propostas no Capítulo 4, e como resultado foi possível ver que a Arquitetura utilizando padrões de Microsserviços se demonstrou mais eficiente, onde a mesma atendeu três vezes mais requisições que a Monolítica, em um tempo de resposta três vezes menor. Mostrando assim que a Arquitetura conseguiu ser mais eficiente e rápida que comparativamente à outra Arquitetura.

A Arquitetura de Microsserviços com padrões permite além disso, uma tolerância a falhas graças, principalmente, ao padrão de disjuntor que permite uma continuidade parcial do serviço, ao qual a Monolítica não consegue lidar tão bem com as falhas, em que as mesmas ocasionam em geral a indisponibilidade de todo o sistema.

Outro fator a notar na Arquitetura de Microsserviços com seus padrões foi a possibilidade em ter uma aplicação com uma melhor escalabilidade e uma melhor manutenibilidade do que a Arquitetura Monolítica.

Essas vantagens descritas se devem principalmente a utilização dos padrões de Microsserviços, sendo eles os precursores da possibilidade de todas essas vantagens descritas anteriormente. Em geral tudo o que foi descrito anteriormente como vantagem, se relaciona com algum padrão, como a questão de atender mais requisições temos o *API Gateway* que coreografa as requisições e as distribui conforme a utilização das máquinas e o padrão de disjuntor já citado que permite a continuidade parcial do serviço. Esses são alguns exemplos, que demonstram que ao utilizarmos padrões conseguimos criar uma Arquitetura com Microsserviços bem mais vantajosa.

O fator em que a Arquitetura de Microsserviços se demonstrou com pior resultado foi na questão de custos, onde se demonstrou aproximadamente três vezes mais cara que a outra Arquitetura. Tal fato já era previsto, levando em conta que a Arquitetura de Microsserviços utilizava mais serviços de nuvem do que a Arquitetura Monolítica.

Algo que não foi avaliado diretamente, mas que aqui cabe ressaltar é a questão da complexidade em relação ao tempo para se desenvolver cada Arquitetura. É importante frisar que os desenvolvedores não tinham experiência prévia com o uso da AWS. Desta forma, na construção das Arquiteturas na nuvem, a Arquitetura Monolítica teve a necessidade de aproximadamente 40 minutos para poder iniciar o seu serviço. Comparativamente, a Arquitetura com padrões de Microsserviços levou 2 horas e 30 minutos para ter seus serviços iniciados. Esses valores são referentes ao tempo necessário para criação, implementação e validação do funcionamento dos serviços na nuvem. Outros serviços associados principalmente com a Arquitetura de Microsserviços com padrões, como questões de sincronização de serviços, controle da fila de mensagens e orquestração com outros serviços, geram também um tempo bem maior de desenvolvimento comparativamente à Arquitetura Monolítica.

Outro ponto a se considerar é a necessidade de haver desenvolvedores com maior experiência para desenvolver tal Arquitetura, o que consequentemente gera também um custo maior no desenvolvimento da aplicação, tendo em vista esta necessidade de uma mão de obra mais qualificada. Este ponto foi o que mais se demonstrou desvantajoso em se utilizar Microsserviços no estudo de Taibi *et al.* (2017).

Novamente, ressalta-se que o cenário criado fora de dimensões enormes, tendo como comparação um site com porte mundial e com grandes acessos diários. Os custos

ficam bem menores em aplicações com um porte menor.

Podemos ver assim um cenário onde é possível ter uma escalabilidade melhor com a Arquitetura de padrões de Microsserviços, permitindo com que aqueles que a adotem possam iniciar suas aplicações e terem seu custo escalável conforme o aumento da demanda dos serviços, fato esse que não é tão simples comparada a Arquitetura Monolítica.

Outro ponto chave seria a questão de poder escalar apenas um serviço, como exemplo o que tiver mais acesso, sem ter a necessidade de aumentar a capacidade computacional de todo o sistema, sendo esse o caso necessário na Arquitetura Monolítica.

Vale ressaltar como a questão de tolerância às falhas na Arquitetura de Microsserviços com padrões é algo que se destaca comparativamente a outra Arquitetura. Ao pensarmos que alguma parte do serviço deixe de funcionar, e o mesmo continue tendo continuidade em outros serviços, podemos pensar que aqueles que não tem esta tolerância, podem ter problemas quanto a isso. Considerando alguns exemplos como lojas de *e-commerce* ou instituições financeiras, o custo em se ter seus serviços indisponibilizados, pode em geral, acarreta em uma enorme perda financeira, e em alguns casos até a dissidência de clientes sobre a organização, gerando além de um problema financeiro, um marketing extremamente negativo sobre a mesma.

O próximo capítulo irá tratar sobre os trabalhos futuros, como melhorias que podem diminuir o custo da Arquitetura de padrões de Microsserviços por meio de algumas estratégias de desenvolvimento de software, alguns pontos que podem ser melhorados futuramente e as considerações gerais sobre o trabalho.

CAPÍTULO 6

Considerações finais e Trabalhos futuros

Neste trabalho vimos que existem vários padrões de Microsserviços, sendo descritos alguns mais a fundo no desenvolvimento dos mesmos. Também foi demonstrado algumas ferramentas para o desenvolvimento dos Microsserviços e um comparativo textual das Arquiteturas: Monolítica, SOA e de Microsserviços.

Depois de todo esse levantamento inicial foi desenvolvida uma aplicação utilizando os padrões de Microsserviços mais comuns encontrados, e ao submeter essa aplicação a avaliações e testes, a mesma acabou por mostrar bem mais vantajosa comparada a uma Arquitetura Tradicional Monolítica. Tal vantagem se deu principalmente na questão de escalabilidade, na disponibilidade, atendendo a mais requisições e também na melhor manutenibilidade da Arquitetura.

Em contrapartida, temos questões como a necessidade de haver um bom conhecimento e estudo prévio dos padrões e da Arquitetura de Microsserviços, fazendo com que haja a necessidade de profissionais com uma qualificação e experiência maior do que aqueles que são demandados para construção de sistemas em uma Arquitetura Monolítica. Porém, o fator em que se teve a maior disparidade comparada a Arquitetura Monolítica foi a questão do custo em se ter uma Arquitetura de Microsserviços com os seus padrões em um serviço de nuvem.

Boa parte desse custo mais elevado se deu devido às chamadas aos serviços de API, como o AWS ELB e AWS API Gateway.

Se tirarmos esses serviços e fizermos novamente a comparação de custos, veremos que a questão de diferença entre as Arquiteturas fica quase irrisória, sendo ainda um pouco maior para a Arquitetura com padrões de Microsserviços, esse valor já é de se esperar tendo em vista que é alocado mais recursos computacionais para poder construir a Arquitetura com os padrões que foram propostos neste trabalho.

O custo elevado nesses serviços se deve ao fato de que primeiramente a Arquitetura de Microsserviços implementa uma porta única de entrada e também ao fato de que a mesma atende mais requisições que a Arquitetura Monolítica, consequentemente há um maior gasto na parte de requisições. Porém, poderia ser implementado técnicas de armazenamento dos dados (*cache*) para validar se há de fato a necessidade de consultar

o servidor novamente ou se os dados que estão sendo requisitados já estão disponíveis. Ressalta-se aqui que o cálculo do custo foi feito levando em conta o máximo de requisições que cada serviço conseguiu atender durante 24 horas por dia em um período de 30 dias, sendo esse fato não usual em aplicações.

Pode-se também pensar em recriar a Arquitetura de Microsserviços, mas agora com a utilização de *Serverless*, em que o custo e a escalabilidade ficariam geridos automaticamente pelo serviço de nuvem bastando apenas ser necessário adicionarmos os códigos da aplicação. Com esse autogerenciamento feito pelos provedores, a questão de custo tende a diminuir bastante no que diz respeito a questão de processamento e de requisições na aplicação, levando em conta que o provedor irá buscar a melhor forma de otimizar a aplicação.

Apesar de tudo que foi elencado podemos considerar aqui que a Arquitetura com padrões de Microsserviços foi bem mais vantajosa e que ao aplicar-se algumas melhorias a desvantagem de custo, que foi a maior discrepância encontrada, pode ser reduzida bastante, deixando a Arquitetura com padrões de Microsserviços mais atraente aos desenvolvedores de software, que visam desenvolver aplicações altamente escaláveis.

Assim, após o trabalho também surgiram alguns pontos bastante relevantes que podem ser explorados de forma mais detalhada. Sendo estes referidos sumariamente aqueles que poderão vir a ser objetos de futuros trabalhos:

- Propor uma arquitetura *Serverless*, e refazer as comparações realizadas no estudo.
- Realizar otimizações na aplicação visando melhorar mais ainda o desempenho da Arquitetura de Microsserviços.
- Implementar técnicas de programação que visem reduzir o custo da Arquitetura de Microsserviços.
- Desenvolvimento de um manual de construção de software usando a Arquitetura de Microsserviços com seus padrões.
- Realizar um estudo mais a fundo sobre cada padrão proposto.
- Desenvolver a Arquitetura de Microsserviços com os padrões sem a utilização de serviços de nuvem.

Referências Bibliográficas

Akulut, A.; Perros, H. G. Performance analysis of microservice design patterns. **IEEE Internet Computing**, v. 23, n. 6, p. 19–27, 2019. 44

ALEXANDER, Christopher. **A pattern language: towns, buildings, construction.** [S.l.]: Oxford university press, 1977. 16, 29

ANDERSON, Charles. **Docker [software engineering].** maio-junho 2015. Disponível em: <https://ieeexplore.ieee.org/abstract/document/7093032>. 13, 23, 24

ARCITURA. **Microservice and Containerization Patterns.**

<Https://patterns.arcitura.com/microservice-patterns>. 30

BATISTA, Felipe de Andrade. Arquitetura de microserviços: Uma solução leve para grandes sistemas no futuro. **Anais do Encontro Nacional de Pós Graduação**, v. 2, n. 1, p. 410–414, 2018. 19, 22

BUSCHMANN, F; MEUNIER, R; ROHNERT, H; SOMMERLAND, P; STAL, M. **Software Patterns.** [S.l.]: John Wiley & Sons, 1996. 29

CHAPPELL, David *et al.* **Introducing windows azure.** 2009. Disponível em: https://www.idt-inc.com/wp-content/uploads/sites/4755/2017/05/IntroducingWindowsAzureFinal_5_5_2011_9_55_46_AM.pdf. 24

DIVIDIR um aplicativo monolítico em microsserviços. 2018. Disponível em: Https://aws.amazon.com/pt/getting-started/projects/break-monolith-app-microservices-ecs-docker-ec2/?trk=gs_card. 35, 36

DUARTE, Luiz. **Design Patterns for Microservices.** julho 2017. Disponível em: <Https://www.luiztools.com.br/post/microservices-vs-soa-entenda-as-diferencias/>. 25

GHOFRANI, Javad; LÜBKE, daniel. Challenges of microservices architecture: A survey on the state of the practice. 05 2018. 29

JOHNSON, Rod; HOELLER, Juergen; DONALD, Keith; SAMPALEANU, Colin; HARROP, Rob; RISBERG, Thomas; ARENDSEN, Alef; DAVISON, Darren; KOPYLENKO, Dmitriy; POLLACK, Mark *et al.* The spring framework—reference documentation. **interface**, v. 21, p. 27, 2004. 23

NETO M. P. M; AUGUSTO, V. S. S. **Tabela de padrões separados por suas aplicabilidade**. 2019. Disponível em: https://docs.google.com/spreadsheets/d/1jtyPTBBBE0o0QihStIY2xoOF81LQ_cN5oZ0i8IdnnKc/edit?usp=sharing. 30

NEWMAN, Sam. **Building Microservices**. 1st. ed. [S.l.]: O'Reilly Media, Inc., 2015. 2 p. ISBN 1491950358, 9781491950357. 16

O'GRADY, Stephen. The difference between soa and microservices isn't size. 07 2017. 26

RICHARDSON, Chris. **A pattern language for microservices**. <Https://microservices.io/patterns>. 30

_____. **What are microservices?** <Https://microservices.io/>. 16

_____. **Microservices patterns: With examples in Java**. [S.l.]: Manning Publications, Shelter Island, 2019. 520 p. 26, 27

SAMPAIO, Cleuton. **SOA e Web services em Java**. [S.l.]: Brasport, 2006. 20, 21

TAIBI, Davide; LENARDUZZI, Valentina; PAHL, Claus. Architectural patterns for microservices: A systematic mapping study. 03 2018. 30

TAIBI, Davide; LENARDUZZI, Valentina; PAHL, Claus; JANES, Andrea. Microservices in agile software development: a workshop-based study into issues, advantages, and disadvantages. p. 1–5, 05 2017. 22, 51

URDANETA, Guido; PIERRE, Guillaume; STEEN, Maarten van. Wikipedia workload analysis for decentralized hosting. **Elsevier Computer Networks**, v. 53, n. 11, p. 1830–1845, July 2009. http://www.globule.org/publi/WWADH_comnet2009.html. 44

VERAS, Manoel. **Arquitetura de Nuvem (AWS): Amazon Web Services**. 2013. Disponível em:

https://books.google.com.br/books?hl=pt-BR&lr=lang_pt&id=RY7kp2dT0jwC&oi=fnd&pg=PA1&dq=amazon+aws&ots=jd-1xhc32R&sig=LYL4r0vnvPjjv8mEbMrf2wm_4m0&redir_esc=y#v=onepage&q=amazon20aws&f=false. 24

WEBER, Taisy Silva. Um roteiro para exploração dos conceitos básicos de tolerância a falhas. **Relatório técnico, Instituto de Informática UFRGS**, 2002. 48

WEISSMANN, Henrique Lobo. **Vire o jogo com Spring Framework**. 2014. Disponível em: https://books.google.com.br/books?hl=pt-BR&lr=lang_pt&id=nm2CCwAAQBAJ&oi=fnd&pg=PP7&dq=spring+framework&ots=ugdVfZ59Dc&sig=cyWRnjbZQ0iT39-5MjU23PeTFYg&redir_esc=y#v=onepage&q=spring20framework&f=false. 13, 23