Avaliação de Desempenho de *API Gateways* para o Gerenciamento de Acessos a Microsserviços

Rodrigo Cargnelutti[1,2], Giulliano Minuzzi[1,2], Everton de Vargas Agilar[2]
Fábio Paulo Basso[1], Maicon Bernardino[1]

[1] Universidade Federal do Pampa (UNIPAMPA), Alegrete, RS, Brasil
[2]Universidade Federal de Santa Maria (UFSM), Santa Maria, RS, Brasil

{rodrigocargnelutti.aluno,fabiobasso}@unipampa.edu.br
gminuzzi@ufsm.br, evertonagilar@ufsm.br, bernardino@acm.org

Abstract. *This study develops and evaluates API Gateways for access control in a microservices architecture. The work compares the developed API Gateway with the Kong API Gateway, using the Locust tool for performance testing with workloads of 100 and 500 simultaneous users. The Mobile and Webservice endpoints were analyzed, with metrics such as the number of requests, response time, and resource usage. The results show that both solutions are efficient, with Kong excelling in network usage, while the developed API Gateway demonstrated better memory consumption. It is concluded that an API Gateways is essential for access control and security in microservices architecture.*

Resumo. *Este estudo desenvolve e avalia API Gateways para o controle de acessos em uma arquitetura de microsserviços. O trabalho compara a API Gateways desenvolvida e o Kong API Gateway, utilizando a ferramenta Locust para testes de desempenho com workloads de 100 e 500 usuários simultâneos. Foram analisados os endpoints Mobile e Webservice, com métricas de número de requisições, tempo de resposta e uso de recursos computacionais. Os resultados mostram que ambas as soluções são eficientes, com o Kong se destacando no uso de rede, enquanto a API Gateways desenvolvida demonstrou melhor consumo de memória. Conclui-se que uma API Gateways é fundamental para o controle de acessos e a segurança na arquitetura de microsserviços.*

1. Introdução

Atualmente, o Centro de Processamento de Dados (CPD) da Universidade Federal de Santa Maria (UFSM) desenvolve e mantém os sistemas institucionais: Sistema de Informações para o Ensino (SIE), Sítios Web e o aplicativo *mobile* UFSM Digital. O CPD enfrenta o desafio de garantir uma comunicação segura entre os *webservices*, essencial para a modernização dos sistemas. Este trabalho propõe a modernização da arquitetura dos sistemas institucionais, com foco no desenvolvimento de uma *API Gateway* para aprimorar a segurança por meio do controle de acesso aos *webservices* do SIE.

A pesquisa foi baseada em uma Revisão Sistemática da Literatura (RSL), que orientou a escolha de tecnologias e ferramentas. Optou-se por desenvolver uma *API Gateway* para centralizar todas as solicitações externas aos *webservices* do SIE, oferecendo controle de acesso e reduzindo os riscos de segurança cibernética relacionados ao acesso não autorizado. Uma *API Gateway* serve como ponto de entrada unificado do sistema e

é um método excelente para aplicar uma camada de segurança [Raj et al. 2023]. A popularidade da arquitetura de microsserviços no desenvolvimento de software tem crescido devido à sua flexibilidade, granularidade e serviços desacoplados [Luz et al. 2018].

O objetivo deste trabalho é centralizar as requisições aos *webservices* do SIE por meio de uma *API Gateway*, aprimorando o controle das interações com sistemas externos e oferecendo uma solução customizada, mais alinhada às demandas específicas da arquitetura dos sistemas da UFSM. Para isso, foram definidos os objetivos específicos: (I) Desenvolver uma *API Gateway* com as funcionalidades para verificação de autenticação, roteamento e registro das requisições; (II) Fornecer um sistema de *logs* para monitoramento e rastreabilidade, fundamental para a segurança e conformidade com a Lei Geral de Proteção de Dados (LGPD); (III) Avaliar o impacto da *API Gateway* no tempo de acesso e desempenho das aplicações que utilizam os *webservices*.

2. Fundamentação Teórica

Arquitetura de Software: é um pilar fundamental, pois define como o software é projetado, mantido e aprimorado ao longo do tempo [De Sordi et al. 2006]. Ela descreve os componentes, suas funções, organização do código, o armazenamento de dados e a comunicação entre as partes. Há dois principais estilos: a arquitetura monolítica, em que o sistema é integrado em um único bloco [Awati and Wigmore 2023], e a arquitetura de microsserviços, que divide o sistema em pequenos serviços independentes [Xiong and Li 2022]. Esta última oferece escalabilidade e elasticidade para infraestrutura do sistema em oposição à arquitetura monolítica [Pasomsup and Limpiyakorn 2021]. Em uma arquitetura de microsserviços, uma *API Gateway* desenvolve um papel muito importante, o de controlar a entrada e a saída de requisições. **Controle de Acesso**: garante que apenas usuários autorizados possam acessar o sistema, protegendo a confidencialidade e integridade dos seus recursos [Majumder et al. 2014]. A segurança em microsserviços é muitas vezes subestimada, ficando a cargo apenas da *API Gateway* a função de proteger os serviços internos [Bhutada and Jyothi 2019]. As melhores estratégias de controle de acesso para microsserviços incluem o uso de sessão distribuída, autenticação por Single Sign-On (SSO), *JSON Web Token* (JWT) no cliente e o uso de uma *API Gateway* [He and Yang 2017]. O controle de acesso é composto por duas partes essenciais: autenticação e autorização. A autenticação confirma a identidade do usuário, enquanto a autorização verifica suas permissões para acessar determinados recursos [Hannousse and Yahiouche 2021]. ***API Gateway***: fornece uma camada de abstração que protege os microsserviços do acesso direto a clientes externos. Uma *API Gateway* é posicionada em um único ponto de contato com clientes externos e ocultam com eficiência os microsserviços de *back-end* de qualquer acesso externo. Em vez de se conectar individualmente em cada microsserviço, o cliente estabelece comunicação apenas com a *API Gateway* [Raj et al. 2023]. A *API Gateway* atua entre os clientes externos e os microsserviços, fornecendo um ambiente de rede privada [Xu et al. 2019]. Na prática, a *API Gateway* é um serviço frequentemente utilizado como intermediador, proporcionando um ponto centralizado para a implementação de políticas de controle de acesso para microsserviços [Preuveneers and Joosen 2019]. ***Kong API Gateway***: É uma *API Gateway* escalável de alto desempenho que implementa funções de roteamento, autenticação, balanceamento de carga e *logging*. O *Kong* funciona como um ponto de entrada central para orquestrar o tráfego de microsserviços [Alencar et al. 2022]. Ele pode atuar na frente de

qualquer tipo de serviço e pode ser estendido por meio de *plug-ins*, com funcionalidades adicionais, como autenticação via *JSON Web Token* (JWT), Controle de Tráfego, *Logging*, entre outros [Xu et al. 2019]. ***Logs***: são registros de informações que documentam eventos de forma cronológica, servindo como uma fonte para consultas e análises futuras [HostMidia 2024]. Eles permitem que os administradores rastreiem o comportamento dos usuários, identifiquem riscos, forneçam uma visão do uso do sistema e ajustem as políticas de acesso. [Fengxuan et al. 2023].

3. Trabalhos Relacionados

Em uma publicação prévia de uma pesquisa de *RSL* [Autores Anônimos 2023], identificamos diversos estudos relacionados à proposta deste trabalho.

O trabalho de [Alencar et al. 2022] propõe uma arquitetura com o *Kong API Gateway*, comparando seu desempenho e segurança ao *X-Road*. Os resultados mostram que o *Kong* apresenta melhor tempo de resposta e sugerem melhorias para ambas ferramentas. No nosso trabalho, implementamos uma *API Gateway* e comparamos seu desempenho com o *Kong*, destacando que ambas as soluções são adequadas, cada uma com vantagens em desempenho e consumo de recursos.

Comparar o desempenho de diferentes soluções para escolher a mais adequada é um processo demorado. Para simplificar essa tarefa, o estudo apresenta o AGE, um serviço que automatiza a implantação de diferentes cenários de *API Gateways* e fornece uma avaliação comparativa de desempenho, permitindo testes mais rápidos e eficientes em diversos ambientes [Moreira et al. 2023]. A nossa pesquisa utiliza a ferramenta *Locust* para avaliar o desempenho entre a *API Gateway* desenvolvida e o *Kong*. Realizamos uma análise comparativa considerando diferentes combinações de *endpoints*, volumes de cargas de trabalho (usuários simultâneos) e cenários de teste.

O *Anser-Gateway* é uma *API Gateway* de alto desempenho em *PHP*, criado para simplificar a complexidade dos microsserviços. Com uma arquitetura baseada em *event loop* e *coroutines*, o *API Gateway* otimiza o processamento de requisições ao utilizar o escalonamento cooperativo. Comparado a outras soluções, essa abordagem oferece maior eficiência, resultando em um desempenho superior de *throughput* [Lee and Tsai 2024]. Em nosso estudo, desenvolvemos uma *API Gateway* usando o *Spring Cloud Gateway*, testamos em cenários reais e comparamos seu desempenho com o *Kong*.

O estudo de [Zuo et al. 2020] busca otimizar o design de *API Gateways*, focando na persistência em banco de dados e na redução do acoplamento entre módulos para melhorar o desempenho, custo e manutenibilidade. A pesquisa realiza testes de desempenho para validar a eficácia da solução proposta. De forma similar, nosso trabalho também avalia o desempenho de *API Gateways*, mas com foco no gerenciamento de acessos a microsserviços, comparando diferentes soluções para determinar as vantagens de cada uma em termos de eficiência e uso de recursos.

A aplicação de *API Gateways* em sistemas embarcados com microsserviços, com foco na análise de requisitos específicos como baixo consumo de recursos, comunicação em tempo real e comparação de diferentes *API Gateways*, é explorada por [Tomić et al. 2022]. Na nossa pesquisa, desenvolvemos uma *API Gateway* para aprimorar o controle de acesso aos microsserviços, comparando seu desempenho e consumo de recursos com a ferramenta *Kong*.

O projeto de uma *API Gateway* como *middleware* em uma arquitetura *PaaS* para um sistema de laboratório é apresentado por [Oktaria et al. 2021], utilizando uma abordagem *SOA* e testes de desempenho para validar a solução. Similarmente, nosso trabalho avalia o desempenho de *API Gateways* para gerenciar acessos a microsserviços, mas em um sistema de gestão educacional, comparando *Kong* e a solução desenvolvida.

4. Desenvolvimento do Trabalho

Esta seção analisa a implementação de uma *API Gateway* para controle de acesso em microsserviços, abordando a solução desenvolvida, o *Kong API Gateway*, a arquitetura proposta e as tecnologias utilizadas.

4.1. Arquitetura Proposta

A arquitetura implementada para o gerenciamento de acessos é baseada no uso de uma *API Gateway* para centralizar as requisições aos microsserviços e assegurar o controle de acesso aos *webservices* da instituição. A Figura 1, ilustra como funciona a arquitetura utilizando uma *API Gateway* e como ela está integrada com os demais sistemas.

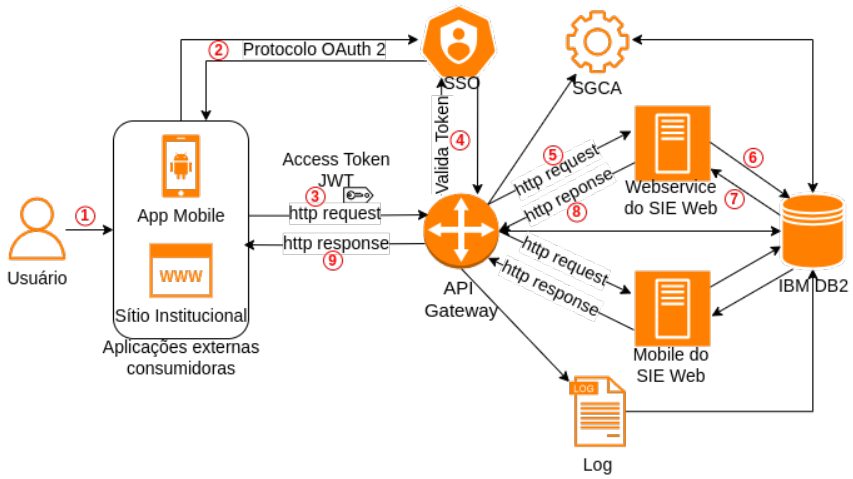

Figura 1. Arquitetura Implementada

O aplicativo *mobile* e os sítios institucionais da UFSM, são as aplicações externas que vão passar a consumir os *webservices* do SIE via *API Gateway*. A *API Gateway* tem o papel de filtrar o tráfego e impedir o acesso não autorizado antes que a solicitação alcance o serviço de *back-end*. Quando o usuário acessa o aplicativo *mobile* ou um site institucional (1), ele é direcionado de forma transparente para a tela de *login* do SSO por meio do protocolo OAuth 2.0 para se autenticar (2). Após a autenticação bem sucedida, o SSO retorna para aplicação um cabeçalho com o campo *authorization*. Este campo *authorization* tem um *token* JWT solicitado pelo usuário na autenticação. Quando uma aplicação externa realiza uma solicitação, ela precisa ser verificada pela *API Gateway* (3), para checar se está autenticada ou não, verificando se no cabeçalho da requisição tem um campo *authorization*, esta verificação é de responsabilidade do SSO (4). Logo, se uma aplicação tenta consumir informações de um *webservices* do SIE via *API Gateway* e o SSO retornar que o *token* é invalido ou expirou sua validade, a *API Gateway* não deixa a solicitação passar e reporta acesso negado (9), por outro lado, se o *token* for validado pelo SSO, então a solicitação terá permissão para prosseguir e consumir o serviço desejado

(5). Dessa maneira, por meio de uma ação do usuário, a aplicação faz uma solicitação que é direcionada pela *API Gateway* para o serviço desejado, um *webservices* do SIE (5), que realiza acesso ao banco de dados (6), o banco de dados por sua vez retorna as informações para o *webservices* do SIE (7), que realiza as operações necessárias para atender à requisição e envia a resposta ao cliente (9) por meio da *API Gateway* (8). A *API Gateway* foi concebida para ser provisionada no *cluster Kubernetes* do CPD.

4.2. Tecnologias Utilizadas

Para desenvolver a *API Gateway* proposta, foi escolhida a tecnologia *Spring Cloud Gateway*, um projeto baseado no *Spring Boot* e *JDK* 21. O *Spring Cloud Gateway* oferece bibliotecas para construir uma *API Gateway* no modelo de programação reativa usando o *Spring WebFlux*, que permite realizar muitas conexões simultâneas de forma eficiente. O *Spring Cloud Gateway* é o sucessor da *API Gateway Zuul* dentro do ecossistema *Spring Boot*, que foi a segunda ferramenta mais mencionada na RSL. Diante dos resultados obtidos na RSL, optou-se por utilizar o *Spring Cloud Gateway* por ser o sucessor da *Zuul* e por sua compatibilidade com o desenvolvimento do SIE. Além disso, a escolha do *Spring Boot* é estratégica, pois facilita a integração com o projeto SSO e com outros sistemas da Instituição que foram desenvolvidos com essa tecnologia. Essa compatibilidade simplifica futuras manutenções e adaptações, com menor custo e esforço, além de ser uma solução familiar para a equipe de desenvolvimento.

4.3. *API Gateway* Proposta

As configurações de serviços e rotas para o *API Gateway* são armazenadas em uma tabela no banco de dados institucional, utilizando o SGBD DB2 para operacionalizar essa funcionalidade. A gestão dos serviços e rotas utilizados pela *API Gateway* foram implementadas no Sistema de Gerenciamento e Controle de Acesso (SGCA) do SIE. Basicamente, a implementação do *API Gateway* consiste de um método que é responsável por configurar as rotas de uma *API Gateway* de forma dinâmica, usando um objeto para definir as rotas com base em uma lista de serviços retornada de uma consulta ao banco de dados. Para cada serviço na lista, são configurados os parâmetros para cada rota. A rota recebe o caminho que deve corresponder, aplica um filtro, adicionando o caminho da rota para o valor especificado e define o endereço de destino para onde a solicitação deve ser encaminhada. O método retorna um conjunto de rotas configurado dinamicamente, ajustando o caminho da requisição e a URI de destino com base nas informações de cada serviço na lista.

Para realizar a verificação do *access token* da solicitação foi utilizado a tecnologia de filtros do *Spring Boot* que permite interceptar a requisição HTTP e verificar se o campo *Authorization* está presente no cabeçalho da solicitação e se é válido. Utilizando a tecnologia de filtros, a *API Gateway* desenvolvida oferece uma camada adicional de segurança, pois consegue filtrar o tráfego antes que ele alcance o serviço de *back-end*. Para rastrear e monitorar as solicitações que chegam a *API Gateway*, foi implementada uma funcionalidade para coletar e registrar as informações dos acessos. Esses registros são armazenados em uma tabela do banco de dados institucional e as informações disponibilizadas em uma interface desenvolvida no portal de SGCA do SIE. Os *logs* gerados pela *API Gateway* permitem o monitoramento das requisições, possibilitando rastrear cada solicitação, identificar sua origem, destino e eventuais erros. Isso é fundamental para: (i) identificar quais

serviços são mais utilizados, por quem, em quais períodos e com que frequência; (ii) detectar comportamentos anômalos, como um aumento repentino no número de requisições, o que pode indicar problemas de performance ou até mesmo um possível ataque.

Dessa forma, a implementação da *API Gateway* com as tecnologias e metodologias descritas proporciona o gerenciamento das rotas e serviços, geração de *logs* e maior controle de acesso aos *webservices* do ecossistema do SIE *Web*. Com isso, assegura-se que os requisitos e objetivos definidos foram plenamente atendidos, além de proporcionar uma base para futuras expansões e melhorias.

5. Avaliação de Desempenho

Para coletar métricas na avaliação de desempenho, usamos a versão 2.25.0 da ferramenta *Locust*[1], utilizada para testes de desempenho em aplicações *web* [Meier 2007]. Projetada para simular um grande volume de usuários acessando um serviço simultaneamente, ela permite avaliar como a aplicação se comporta sob alta demanda. Além disso, oferece a criação de cenários realistas com métricas em tempo real, que podem ser programados por meio de *scripts* em *Python*. Durante a execução do teste, o *Locust* coleta e exibe métricas de desempenho, como tempo de resposta, taxa de requisição por segundo, taxa de sucesso/falha, percentis e erro de requisição. Após o teste, é possível analisar os dados coletados para identificar gargalos, falhas de desempenho e áreas que precisam de otimização. Escolhemos a ferramenta *Locust* para a análise do teste de desempenho devido à sua flexibilidade, simplicidade e por ser de código aberto.

A análise de desempenho desses serviços é motivada pela necessidade de garantir a eficiência dos *webservices* que suportam as operações da instituição. Um dos serviços testados é o *endpoint* `Servidor`[2] do portal de *Webservice*. Este serviço recebe uma solicitação contendo o parâmetro `contrato` com o número de contrato e, em resposta, fornece informações detalhadas sobre o servidor. O segundo serviço submetido aos testes de desempenho foi o *endpoint* `Horários de Ônibus`[3] que fornece informações sobre os horários de ônibus, disponibilizado por meio do portal *mobile*. Os serviços em teste são componentes vitais do ecossistema digital da UFSM, e a análise de seu desempenho garante que eles continuem atendendo às necessidades da instituição de forma eficiente.

Para as avaliações, os sistemas foram configurados em uma máquina virtual instanciada na nuvem privada do CPD, como segue: 40 GB de memória RAM, 12 núcleos de processamento, 150 GB de armazenamento em disco e o SO Linux Debian 11 (Bullseye). O SIE, a *API Gateway* desenvolvida e o *Kong API Gateway* são executados em contêineres utilizando a versão 26.1.4 do *Docker*. Vale destacar que os sistemas são executados de forma isolada em seus respectivos contêineres, não ocorrendo competição por recursos computacionais. O escopo foi definido em dois cenários `C1` e `C2` para avaliar a eficácia das *API Gateways* na arquitetura proposta para os *webservices* institucionais: **C1:** Efetuar as requisições utilizando a *API Gateway* desenvolvida neste trabalho; **C2:** Realizar as mesmas requisições por meio da ferramenta *Kong*. Para as avaliações, criamos *Tasks* e definimos o comportamento dos usuários que farão as requisições para cada cenário e *endpoint* por meio de *scripts* em *Python*.

[1] `https://locust.io`
[2] `https://portal.ufsm.br/webservice/ws/site/servidor/servidor.json`
[3] `https://portal.ufsm.br/mobile/webservice/flutter/horariosOnibus`

As avaliações foram realizadas de forma segmentada por Cargas de Trabalho (*workloads*): 100 (W1) e 500 (W2) usuários simultâneos, cenários (C1 e C2) e *endpoints* (``/horarioOnibus'' - portal *Mobile* e ``/servidor'' - portal *Webservice*). Destacamos que os *workloads* escolhidos refletem cenários reais de uso dos sistemas da UFSM. Determinamos que todos os usuários serão gerados em um período de 120 segundos. A taxa de geração de usuários por segundos (*Ramp Up Time*) foi calculada dividindo a quantidade total de usuários pelo tempo para que todos os usuários entrem em operação. Todas as execuções das cargas de trabalho foram avaliadas por um período de 60 minutos.

A primeira avaliação A1 foi realizada com uma carga de trabalho (*workload*) de 100 usuários simultâneos (W1), uma taxa de geração de 0,83 usuários por segundo (*Ramp Up Users*), o mesmo se repete para as avaliações A2, A5 e A6, sendo as duas últimas para o *Webservice*. Por sua vez, a avaliação A3 foi executada com uma carga de 500 usuários simultâneos (W2), gerados a uma taxa de 4,17 usuários por segundo, repetindo o mesmo comportamento para as avaliações A4, A7 e A8, sendo as duas últimas para o *Webservice*.

Coletamos as métricas sobre o Número Total de Requisições (*Requests*), Número Total de Falhas (*Fails*), Tempo Médio de Resposta (*Average*), Taxa de Requisições por Segundo (*Current RPS*) e a Taxa de Falhas por Segundo (*Current Failures/s*). Os resultados são apresentados em uma tabela para cada *workload* e *endpoint* para os dois cenários.

Tabela 1. Avaliações de desempenho realizadas com as *APIs*.

Avaliação	Endpoint	Worload	Cenário	#Requisições	#Falhas	Média/s	Requisições/s	Falhas/s
A1	**Mobile**	**W1**	**C1**	100.859	12.867	2,00	28,02	3,57
A2			**C2**	104.658	15.909	1,88	29,07	4,42
A3		**W2**	**C1**	261.799	119.097	5,25	72,72	33,08
A4			**C2**	266.643	112.615	5,12	74,07	31,28
A5	**Webservice**	**W1**	**C1**	224.785	0	0,07	62,44	0
A6			**C2**	225.160	0	0,07	62,54	0
A7		**W2**	**C1**	532.798	411	1,82	148,00	0,11
A8			**C2**	557.121	11	1,67	154,76	0

A análise de desempenho das *APIs* nos cenários C1 (*API Gateway* desenvolvida) e C2 (*Kong*) foi realizada para os *endpoints Mobile* e *Webservice*, utilizando as cargas de trabalho W1 e W2. As métricas analisadas incluem o número total de requisições, falhas, tempo médio de resposta, Requisições Por Segundo (RPS) e falhas por segundo.

Endpoint Mobile: Para o W1, 100 usuários simultâneos no cenário C1, o sistema processou 100.859 requisições com uma taxa de 28,02 RPS e um tempo médio de resposta de 2,00 segundos. No entanto, a taxa de falhas foi significativa, com 12.867 falhas, resultando em 3,57 falhas por segundo. No C2, o número de requisições aumentou para 104.658, com uma taxa de 29,07 RPS e um tempo médio de resposta levemente menor de 1,88 segundos. O número de falhas também aumentou, totalizando 15.909, o que gerou 4,42 falhas por segundo. Embora o *Kong* tenha processado mais requisições, houve uma desvantagem em termos de falhas. Já para o W2, 500 usuários, com uma carga de trabalho maior, o cenário C1 processou 261.799 requisições, com uma taxa de 72,72 RPS, mas com 119.097 falhas, resultando em 33,08 falhas por segundo e um tempo médio de resposta de 5,25 segundos. No cenário C2, o número total de requisições foi um pouco maior, 266.643, com uma taxa de 74,07 RPS e um tempo de resposta melhorado de 5,12 segundos. O número de falhas reduziu para 112.615, resultando em uma taxa de falhas de 31,28 por segundo. Nesse cenário C2, o *Kong* foi mais eficiente em processar requisições,

com uma leve redução no número de falhas em comparação à *API Gateway* desenvolvida. O cenário C2 é uma alternativa, com tempos de resposta rápidos e um RPS elevado, apesar no significativo número de falhas registrados. Por questões de limitações de espaço, disponibilizamos os gráficos das avaliações no repositório deste estudo[4].

Endpoint Webservice: Para W1 no cenário C1, o *Webservice* processou 224.785 requisições com uma taxa de 62,44 RPS e um tempo médio de resposta de 0,07 segundos, sem registrar falhas. No cenário C2, o número de requisições foi quase idêntico, com 225.160 requisições e uma taxa de 62,54 RPS. O tempo médio de resposta permaneceu em 0,07 segundos, e também não houve falhas. Ambos os cenários mostraram desempenho equilibrado com 100 usuários simultâneos e sem erros. Por sua vez no *workload* W2 (500 usuários simultâneos) e no C1, foram processadas 532.798 requisições, com uma taxa de 148 RPS e um tempo médio de resposta de 1,82 segundos. Foram registradas 411 falhas, com uma taxa de 0,11 falhas por segundo. No (C2), o número de requisições aumentou para 557.121, com uma taxa de 154,76 RPS e um tempo médio de resposta menor, de 1,67 segundos. O número de falhas reduziu para 11 requisições, resultando em uma taxa de falhas por segundo próxima de zero, indicando que o *Kong* oferece um bom desempenho. As diferenças de desempenho entre os dois cenários são mínimas no que diz respeito à taxa de requisições, porém o *Kong* se destaca por ser mais eficiente em termos de tempo de resposta com menos falhas.

Além das métricas de desempenho dos acessos aos *endpoints*, também foram coletadas métricas de consumo de recursos computacionais, como `Memória`, `Processamento` e `Tráfego de Rede`. Em termos de consumo de memória, a *API Gateway* desenvolvida (C1) demonstrou ser mais eficiente, utilizando cerca de 50% menos memória do que o *Kong* (C2). Esse comportamento se manteve para as duas cargas de trabalho (W1 e W2). Essa diferença representa uma vantagem no uso de recursos computacionais, permitindo um uso mais eficiente da memória disponível.

O consumo de *Central Processing Unit* (CPU) permaneceu equilibrado entre os dois cenários, com variações mínimas que não impactam significativamente a escolha de uma solução sobre a outra. Ambas *API Gateways* mostraram um uso eficiente de CPU, consumindo em torno de 10% para (W1), e aumentando para 15% em (W2) para o *endpoint Mobile*. No entanto, no *endpoint Webservice* sob (W2), (C1) chegou a consumir 40% de CPU em comparação com os 20% consumidos pelo (C2). Apesar dessa diferença, o consumo de CPU demonstra que ambas as soluções tem consumo equilibrado, mas o *Kong* pode ter uma leve vantagem em cenários de alta demanda.

No quesito tráfego de rede, o *Kong* (C2) apresentou uma vantagem clara, consumindo, menos tráfego de rede, tanto no recebimento quanto no envio de dados, em comparação com a *API Gateway* desenvolvida (C1). Para o *endpoint Mobile* sob (W2), o *Kong* (C2) teve um tráfego médio aproximado de rede de 4,9 GB de dados recebido (RX) e 5,5 GB enviado (TX), comparado aos 13,15 GB recebido (RX) e 10 GB enviado (TX) pela *API Gateway* desenvolvida (C1). No *endpoint Webservice*, com o *Kong* (C2) consumindo um tráfego médio aproximado de rede de 5,9 GB de dados recebidos (RX) e 7 GB enviados (TX), comparado aos 16,1 GB recebidos (RX) e 12,1 GB enviados (TX) pela *API Gateway* desenvolvida (C1).

[4]Repositório Zenodo: `https://doi.org/10.5281/zenodo.13776796`

A análise comparativa entre os cenários revela que a solução desenvolvida se destaca em termos de consumo de memória, enquanto o *Kong* apresenta melhor desempenho no uso de rede, especialmente sob cargas mais pesadas, provavelmente devido à compressão de dados, que reduz o tráfego. O consumo de CPU se manteve equilibrado entre as duas soluções, sugerindo que ambas as *gateways* são eficientes no processamento das requisições. Essas observações fornecem *insights* valiosos para a escolha da *API Gateway* a ser utilizada, dependendo das prioridades de recursos computacionais e da infraestrutura disponível. As métricas de consumo de recursos computacionais foram coletadas diretamente do contêiner de cada *API Gateway* (C1 e C2) utilizando a *Portainer Community Edition*[5], versão 2.20.3. Uma ferramenta de código aberto que oferece uma maneira simples de gerenciar contêineres *Docker* e também fornece uma funcionalidade sobre o consumo de "Memória", "Processamento" e "*Network*" em tempo real sobre o contêiner.

6. Considerações Finais

No cenário em constante evolução do desenvolvimento de software, a arquitetura de microsserviços tornou-se a base para a construção de sistemas modernos, robustos e escaláveis. Uma *API Gateway*, como componente central, desempenha um papel fundamental na comunicação, aprimorando o controle de acesso e a segurança de um ecossistema com base em microsserviços. O trabalho proposto desenvolveu e avaliou sistemas de *API Gateways* para controle de acesso aos *Websevices* do SIE da UFSM, com o objetivo de possibilitar uma comunicação mais segura com sistemas externos. Nas avaliações de desempenho ficou claro que o uso de uma *API Gateway* se justifica nessa arquitetura, trazendo vantagens como controle de acesso, roteamento e segurança. A comparação entre a *API Gateway* desenvolvida e o *Kong API Gateway* mostrou que ambas são soluções adequadas, cada uma com suas vantagens em termos de consumo de recursos e desempenho. O estudo oferece uma contribuição importante ao propor uma solução de *API Gateway* que atende aos desafios atuais de controle de acesso na arquitetura de microsserviços, com potencial para evoluir e atender às futuras necessidades da UFSM.

Agradecimentos

Os autores agradecem à FAPERGS (Projeto 22/2551-0000841-0) pelo apoio ao trabalho.

Referências

Alencar, J., Magalhães, R., Vasconcelos, D., Chaves, S., Oliveira, J., Bessa, A., and Rodrigues, E. (2022). Comparação de desempenho entre soluções de interoperabilidade. In *X Workshop de Computação Aplicada em Governo Eletrônico*, pages 25–36. SBC.

Autores Anônimos (2023). Titulo Omitido para Avaliação às Cegas. pages 228–237.

Awati, R. and Wigmore, I. (2023). Monolithic architecture.

Bhutada, S. and Jyothi, K. (2019). Enhancing security to the MicroService (MS) architecture by implementing Authentication and Authorization (AA) service using Docker and Kubernetes. *Int. Journal of Innovative Tech. and Exploring Eng.*, 8(6):401–407.

De Sordi, J. O., Marinho, B. d. L., and Nagy, M. (2006). Benefícios da arquitetura de software orientada a serviços para as empresas: análise da experiência do abn amro brasil. *Journal of Information Systems and Technology Management*.

[5]Portainer: `https://www.portainer.io`

Fengxuan, W., Li, Z., Yancheng, Y., Xinzheng, Z., Gang, S., and Yong, G. (2023). Research on service security reinforcement scheme based on application gateway. In *3rd Int. Conf. on Data Science and Computer Application*, pages 113–116. IEEE.

Hannousse, A. and Yahiouche, S. (2021). Securing microservices and microservice architectures: A systematic mapping study. *Computer Science Review*, 41.

He, X. and Yang, X. (2017). Authentication and authorization of end user in microservice architecture. *Journal of Physics: Conference Series*, 910(1).

HostMidia (2024). Logs do sistema operacional – o que são e para que servem?

Lee, W.-T. and Tsai, M.-K. (2024). Implementing a php api gateway based on microservices architecture. In *48th Annual Computers, Software, and Applications Conference*, pages 1578–1579. IEEE.

Luz, W., Agilar, E., de Oliveira, M. C., de Melo, C. E. R., Pinto, G., and Bonifácio, R. (2018). An experience report on the adoption of microservices in three brazilian government institutions. In *XXXII Brazilian Symp. on Soft. Eng.*, pages 32—-41. ACM.

Majumder, A., Namasudra, S., and Nath, S. (2014). *Taxonomy and Classification of Access Control Models for Cloud Environments*, pages 23–53.

Meier, J. (2007). *Performance Testing Guidance for Web Applications: Patterns & Practices*. ITPro collection. Microsoft Press.

Moreira, P., Ribeiro, A., and Silva, J. M. (2023). AGE: Automatic Performance Evaluation of API Gateways. In *Symposium on Computers and Communications*, pages 405–410. IEEE.

Oktaria, D., Ginting, J. A. M. K., Abdurohman, M., and Yasirandi, R. (2021). Design of api gateway as middleware on platform as a service. *Indonesia Journal on Computing (Indo-JC)*, 6(3):47–62.

Pasomsup, C. and Limpiyakorn, Y. (2021). *HT-RBAC: A Design of Role-based Access Control Model for Microservice Security Manager*, pages 177–181.

Preuveneers, D. and Joosen, W. (2019). Towards Multi-party Policy-based Access Control in Federations of Cloud and Edge Microservices. In *European Symposium on Security and Privacy Workshops (EuroS&PW)*, pages 29–38. IEEE.

Raj, P., Vanga, S., and Chaudhary, A. (2023). *Microservices Security: The Concerns and the Solution Approaches*, pages 289–298. Wiley-IEEE Press.

Tomić, M., Dimitrieski, V., Vještica, M., Župunski, R., Jeremić, A., and Kaufmann, H. (2022). Towards applying api gateway to support microservice architectures for embedded systems.

Xiong, Q. and Li, W. (2022). Design and Implementation of Microservices Gateway Based on Spring Cloud Zuul. In *3rd Int. Conf. on Computer Information and Big Data Applications*, pages 1–5. IEEE.

Xu, R., Jin, W., and Kim, D. (2019). Microservice security agent based on api gateway in edge computing. *Sensors (Switzerland)*, 19(22).

Zuo, X., Su, Y., Wang, Q., and Xie, Y. (2020). An api gateway design strategy optimized for persistence and coupling. *Advances in Engineering Software*, 148:102878.