Avaliação da Escalabilidade da Plataforma FIWARE em Ambientes Contêinerizados

Pedro M. Mujica[1], **Marcio S. Oyamada**[1]

[1]Universidade Estadual do Oeste do Paraná (Unioeste) – Cascavel, Brazil

{pedro.mujica, marcio.oyamada}@unioeste.br

Abstract. *Smart City platforms are designed to store data and provide access, enabling the integration of various protocols and devices. The FIWARE platform offers components for storing and visualizing contextual data. However, evaluating its performance under different load scenarios is essential to ensure quality and scalability. This work aims to evaluate the scalability of the FIWARE platform in data ingestion scenarios involving IoT devices. In this study, a scenario with devices connected to LoRaWAN networks is considered, where devices are simulated by sending messages via the MQTT (Message Queuing Telemetry Transport) protocol.*

Resumo. *Plataformas de Cidades Inteligentes visam armazenar e permitir acesso aos dados fornecendo a integração entre diferentes protocolos e dispositivos. A plataforma FIWARE oferece componentes para armazenamento e visualização de dados contextuais. No entanto, avaliar seu comportamento sob diferentes cenários de carga é essencial para garantir sua qualidade e escalabilidade. Este trabalho tem como objetivo avaliar a escalabilidade da plataforma FIWARE em cenários de ingestão de dados provenientes de dispositivos IoT. Neste estudo, considera-se um cenário com dispositivos conectados a redes LoRaWAN, simulados por meio do envio de mensagens via protocolo MQTT (Message Queuing Telemetry Transport).*

1. Introdução

O espaço urbano é um ambiente dinâmico de atividades sociais, econômicas e políticas, cujo crescimento deve ser acompanhado por uma gestão eficiente. Dada a intensificação do crescimento populacional, a gestão inadequada de recursos pode gerar diversos desafios que comprometem a sustentabilidade e a funcionalidade dos centros urbanos [Monzón 2015]. Ademais, esse cresimento urbano frequentemente não é planejado, contribuindo para o surgimento de problemas sociais, como degradação ambiental, violência, serviços de saúde ineficientes e baixa qualidade na prestação de serviços públicos [Wenge et al. 2014].

Dadas as consequências da expansão urbana, a infraestrutura demanda ampliação [Zanella et al. 2014]. Diante disso, as Cidades Inteligentes surgem como uma abordagem para mitigar essas dificuldades, por meio da aplicação do conceito de Internet das Coisas (IoT), que impulsiona o desenvolvimento tecnológico por meio do uso de microcontroladores, sensores e outras tecnologias voltadas à gestão urbana. A integração de tecnologias IoT permite que as cidades se tornem mais eficientes, sustentáveis e responsivas às necessidades dos cidadãos [Chan 2021]. Sendo assim, essas tecnologias impulsionam o

desenvolvimento de soluções eficazes para o gerenciamento urbano, com o uso eficiente e sustentável dos recursos disponíveis [Fernández-Añez et al. 2018].

No entanto, implementações IoT também apresentam desafios relacionados ao desempenho no gerenciamento de dispositivos como sensores, veículos e sistemas automatizados que demandam processamento e análise de dados em tempo real. Além disso, a diversidade de protocolos de comunicação intensifica a heterogeneidade tecnológica, comprometendo a interoperabilidade entre dispositivos e sistemas [Panfilis 2021]. Desta forma, as plataformas unificadas de Cidades Inteligentes surgem para viabilizar a convergência destes dados e facilitar o processo de integração, gerenciamento e desenvolvimento de aplicações [Santana et al. 2016]. Apesar disso, diversas aplicações de Cidades Inteligentes enfrentam questões relacionadas à capacidade de escalabilidade e manutenção [de M. Del Esposte et al. 2019], aspectos que precisam ser devidamente testados.

Neste trabalho, adota-se a plataforma unificada FIWARE [FIWARE Foundation 2026] para a avaliação de sua escalabilidade, um requisito não funcional crítico para sua aplicação no contexto urbano, pois à medida que ocorre a expansão das cidades, há um crescimento exponencial de dispositivos e do tráfego de dados. Dessa forma, sistemas e mecanismos de uma Cidade Inteligente devem ser escaláveis [Rizi and Seno 2022], para suportar a infraestrutura urbana e lidar com um grande volume de sensores [Santana et al. 2017].

Este artigo está organizado da seguinte forma. A Seção 2 apresenta a plataforma FIWARE. A Seção 3 descreve os trabalhos relacionados. A Seção 4 apresenta a metodologia de avaliação e a Seção 5 os resultados obtidos. Por fim, a Seção 6 apresenta as conclusões.

2. Plataforma FIWARE

A plataforma surgiu a partir da parceria pública-privada *Future Internet Public-Private Partnership* (FI-PPP), com o objetivo de acelerar o desenvolvimento e a adoção de tecnologias como IoT, Cloud e *Big Data* no contexto Europeu. Atualmente, a gerência do projeto é feita pela *FIWARE Foundation*, uma organização sem fins lucrativos responsável pela manutenção e desenvolvimento de seus componentes. [FIWARE Foundation 2026].

A plataforma FIWARE possui padrões de coleta, armazenamento e publicação de dados, tendo os dados de contexto que representam o estado atual de entidades como sensores, aplicações de tempo real e outros. Além disso, a plataforma também oferece ferramentas de comunicação multiprotocolo e diversidade entre linguagens e protocolos IoT possibilitando a integração facilitada com outros sistemas e soluções em múltiplas redes.

Dentre o vasto catálogo das ferramentas do FIWARE, a classificação ocorre de acordo com 5 principais categorias: ferramentas de *Deployment*, gerenciamento de contexto, processamento, análise e monitoramento de Contexto, Interface para IoT e Robótica, como apresentado na Figura 1.

A arquitetura da plataforma é baseada em componentes desacoplados e distribuídos, permitindo que diferentes serviços sejam implantados de forma independente em ambientes conteinerizados. Essa abordagem favorece características como flexibilidade, interoperabilidade e escalabilidade, possibilitando a integração de múltiplos serviços de

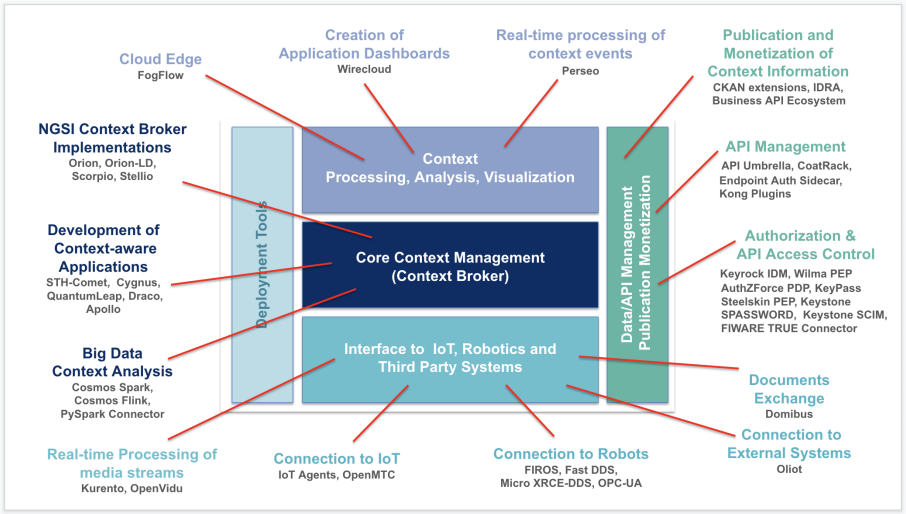

Figura 1. Classificação dos componentes [Panfilis 2021].

armazenamento, processamento e visualização de dados.

Para a integração com dispositivos IoT e diferentes protocolos de comunicação, a plataforma disponibiliza os chamados IoT Agents. Esses componentes atuam como intermediários entre os dispositivos e o Orion Context Broker, realizando a tradução entre protocolos específicos, como MQTT, LoRaWAN e HTTP, para o padrão NGSI utilizado pela plataforma.

3. Trabalhos relacionados

A avaliação da escalabilidade de plataformas de Cidades Inteligentes é fundamental para a garantia do processamento eficiente de dados e o atendimento às demandas dos serviços urbanos. Entretanto, esta temática não é recente na literatura, nesta seção, são apresentados trabalhos que abordam sobre a escalabilidade de plataformas *open-source*, com destaque para a plataforma FIWARE.

Martınez et al. (2016) avaliaram a utilização da plataforma FIWARE no campo da Agricultura de Precisão, propondo uma arquitetura com múltiplos *brokers* baseados no modelo *Publish/Subscribe* e integrada com compenentes de *Big Data*. A análise de escalabilidade foi conduzida por meio do envio de requisições HTTP ao Orion Context Broker, responsável pelo gerenciamento de dados de contexto provenientes de dispositivos IoT. Os resultados indicam que a arquitetura apresenta boa escalabilidade para aplicações em agricultura de precisão, no entanto o trabalho aponta limitações como a complexidade da arquitetura proposta com múltiplos *brokers* e a necessidade de ajustes finos para desempenho como configurações de *Pool* de *threads* e o processamento em lote para persitência dos dados.

Araujo et al. (2019) propõem uma arquitetura baseada nos componentes FIWARE com o uso de IoT *Agents* e implantada em ambiente de nuvem, com o objetivo de avaliar cenários de larga escala utilizando diferentes configurações de instâncias. Para isso, foram simulados dispositivos IoT com a geração de carga utilizando protocolos MQTT e LWM2M baseado no CoAP. Os resultados do estudo identificaram limitações da escalabilidade em nuvem, atribuídos em parte pelo modelo de execução *single-thread* dos IoT *Agents*, bem

como à necessidade de escalonamento do banco de dados MongoDB.

Por último, Domingos et al. (2024) abordam a escalabilidade da plataforma Sentilo, considerando sua aplicação voltada a uma cidade de médio porte. Os testes de carga foram realizados utilizando a plataforma JMeter cujos resultados evidenciam a capacidade da plataforma suportar diferentes taxas de requisições, bem como apresenta capacidade de escalonamento vertical à medida que os recursos de hardware são ampliados.

Tendo como base os outros trabalhos mencionados, o presente estudo compartilha similaridades no que se refere à avaliação da escalabilidade de plataformas de Cidades Inteligentes. No entanto, nos artigos analisados não foram encontrados estudos que abordem a persistência dos dados dos dispositivos simulados em conjunto com o *broker* MQTT. Nesse sentido, a contribuição principal deste trabalho encontra-se na análise da persistência dos dados em testes de carga, pois por meio da persistência, temos os dados que serão disponibilizados para análise, extração de conhecimento e uso pelo usuário final. Adicionalmente, este trabalho avalia o uso da ferramenta MQTT *Shared Subscriptions* [HiveMQ 2024] para avaliação da escalabilidade horizontal. No decorrer desta seção, serão detalhados o procedimento metodológico para a avaliação da escalabilidade, além de ferramentas, arquitetura do projeto e configuração de hardware utilizada.

4. Metodologia

4.1. Arquitetura do projeto

A configuração da plataforma FIWARE envolve a utilização de diferentes componentes para a coleta e o gerenciamento de dados. Desta forma, vários componentes do catalógo do FIWARE foram empregados. O Orion Context Broker[FIWARE 2024c] é o componenete principal para o gerenciamento de informações de contexto em tempo real, pois permite que diferentes sistemas acessem e monitorem o estado de entidades em um ambiente conectado.

A arquitetura de comunicação utilizada no Orion segue o modelo *Publish/Subscribe* em que entidades publicam informações para o *Broker* central e somente assinantes inscritos em determinado tópico recebem as notificações do tópico correspondente. Dessa maneira, o *Context Broker* atua como intermediário para troca de dados entre editores e assinantes. As informações referentes às entidades, inscrições e registros do Broker são armazenadas no Banco de Dados *NoSQL* MongoDB[MongoDB 2024].

Em um cenário real, a integração entre o Orion *Context Broker* e dispositivos LoRaWAN é realizada utilizando servidores de rede, como o The Things Stack ou o ChirpStack, responsáveis pela gestão da comunicação e coleta de dados dos dispositivos. Esses servidores oferecem suporte à integração com o protocolo MQTT, permitindo o encaminhamento dos dados a partir da inscrição nos tópicos que recebem dados dos dispositivos. Nesse contexto, o LoRaWAN IoT Agent [FIWARE 2024b] atua como intermediário entre o servidor MQTT que recebe os dados oriundos de dispositivos LoRaWAN e o Orion Context Broker que posterioriormente disponibiliza os dados de contexto na plataforma.

Contudo, considerando o cenário de testes de carga adotado neste trabalho, não foram empregados servidores de rede LoRaWAN reais. Em seu lugar, foi utilizado o *Eclipse Mosquitto*[Eclipse Foundation 2024], um *broker* MQTT local, com a finalidade

de emular o comportamento de dispositivos LoRaWAN e realizar o envio de dados ao LoRaWAN IoT Agent, que encaminhará os dados para a plataforma.

Como o Orion foi projetado como um *broker*, sendo não adequado para a persistência de grande quantidade de dados devido a limitações de armazenamento, o conector *Cygnus Connector*[FIWARE 2024a] é utilizado para coletar dados históricos do Orion e armazenar estes dados no Banco de Dados PostgreSQL[PostgreSQL Global Development Group 2024] . Na arquitetura utilizada neste trabalho, o Orion é configurado para enviar notificações ao Cygnus sempre que ocorre uma alteração no estado das entidades, como atualizações ou remoções de dados.

A Figura 2 apresenta a organização dos diferentes componentes utilizados neste trabalho. A configuração e comunicação de todos componentes da arquitetura a fim de garantir um ambiente flexível, escalável com fácil implantação, os serviços foram executados em containers Docker , permitindo que cada serviço opere de maneira isolada, evitando conflitos de dependências e simplificando a execução do sistema.

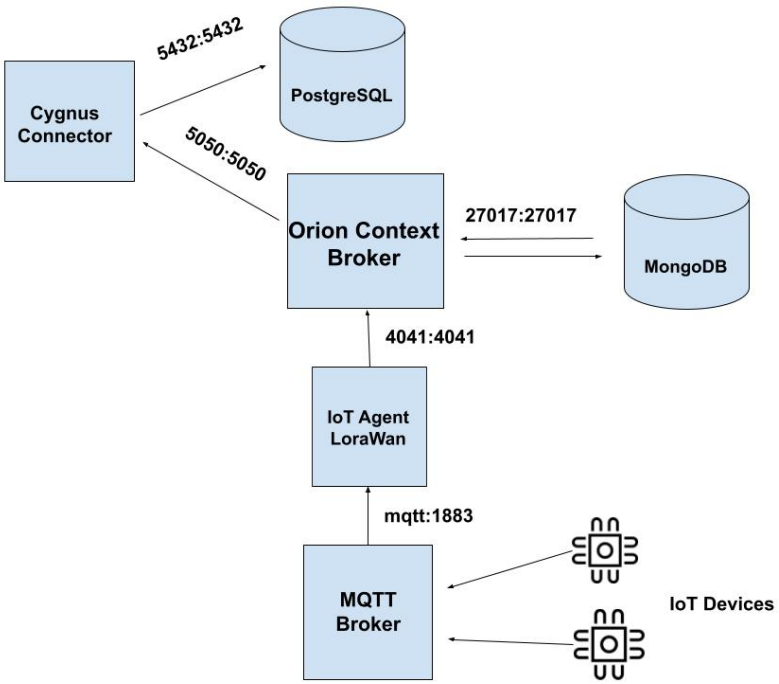

Figura 2. Arquitetura e componentes do Projeto

4.2. Ambiente de Teste

Considerando a arquitetura adotada, os dispositivos simulados são representados por clientes MQTT, que enviam dados para armazenamento no FIWARE. Visando simular vários dispositivos, foi utilizado a ferramenta Apache JMeter, possibilitando a simulação de diferentes taxas de envio. Por meio de um *plugin* específico, foi possível configurar um conjunto de clientes MQTT capazes de enviar pacotes de dados com o conteúdo equivalente a um pacote de um dispositivo conectado via LoRaWAN.

Nos testes realizados, foram configurados 30 *threads* por requisição, em que cada *thread* representa um dispositivo LoRaWAN simulando o envio de 4 atributos: *DateTime*, pressão barométrica, temperatura e umidade relativa. O tempo de *ramp-up*, definido como

o intervalo necessário para atingir o número máximo de *threads* ativas, foi configurado em 20 segundos. Cada teste foi executado por 1 minuto.

Os experimentos foram conduzidos em um ambiente composto por servidor e um computador cliente, responsável pela execução do JMeter. O servidor possui um processador AMD Ryzen 7 5700G 4.67 GHz, com 8 núcleos, além de 64 GB de memória RAM, sendo responsável pela execução da arquitetura FIWARE em contêineres. A máquina cliente, utilizada para a geração de carga, possui um processador AMD Ryzen 7 7730U 4.54 GHz, com 8 núcleos e 16 threads, frequência máxima e 8 GB de memória RAM. A comunicação entre cliente e servidor foi realizada por meio de rede local cabeada, apresentando uma latência média de aproximadamente 5.3 ms, conforme medido por testes de conectividade. A rede utilizada não foi isolada de outros tráfegos provenientes de outros computadores do laboratório.

4.3. Cenários de Teste de Carga

A estratégia para avaliação da escalabilidade é composto por dois cenários de arquitetura para a ingestão dos dados, além de sete diferentes níveis distintos de requisições por segundo.

No primeiro cenário, foi utilizado a arquitetura apresentada na Figura 2, no qual foram realizadas envio de dados com cargas de 15, 100, 200, 300, 400, 500 e 1000 requisições por segundo (RPS), sendo cada configuração executada 5 vezes. Após cada conjunto de testes, o Banco de Dados da plataforma teve suas tabelas deletadas.

No segundo cenário, foi avaliado a escalabilidade horizontal, por meio da utilização de três instâncias do LoRaWAN IoT Agent que foram testadas com os mesmos níveis de requisições por segundo. Para tal fim, foi utilizado o mecanismo MQTT *Shared Subscriptions*, no qual múltiplos clientes compartilham uma única inscrição em um tópico MQTT, assim distribuindo mensagens entre os agentes e promovendo o balanceamento de carga. A Figura 3 apresenta a arquitetura do segundo cenário.

5. Resultados

Os resultados são apresentados com base nas duas arquiteturas distintas: uma sem balanceamento de carga e outra com balanceamento de carga, utilizando três instâncias do LoRaWAN IoT Agent, associadas a três clientes MQTT.

Para a comparação entre os cenários, foram utilizadas as métricas de percentual de escritas concluídas e média de escritas concluídas. Neste trabalho, cada requisição enviada contém quatro atributos distintos (*temperatura*, *umidade*, *pressão barométrica* e *datetime*), resultando em quatro registros persistidos no banco de dados PostgreSQL.

Dessa forma, o número total de escritas realizadas foi calculado a partir da quantidade de linhas persistidas no PostgreSQL, representada pela variável L, dividida pela quantidade de atributos enviados em cada requisição, conforme apresentado na Equação 1.

$$\text{Escritas} = \frac{L}{4} \tag{1}$$

O percentual de escritas concluídas foi calculado considerando a razão entre o número de escritas realizadas e o número esperado de requisições durante o período

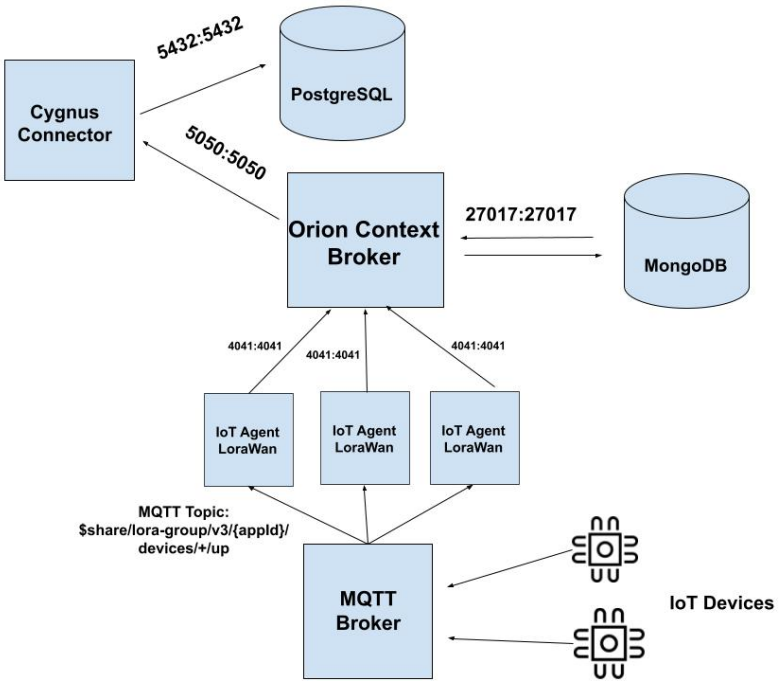

Figura 3. Componentes do segundo cenário

de execução do teste. Para isso, foi utilizada a variável *RPS* (*Requests Per Second*), correspondente às taxas de 15, 100, 200, 300, 400, 500 e 1000 requisições por segundo avaliadas neste trabalho. Considerando que cada teste possui duração de 60 segundos, o percentual de escritas concluídas é definido pela Equação 2.

$$\text{Percentual de Escritas Concluídas} = \frac{\text{Escritas}}{(\text{RPS} \times 60)} \times 100 \tag{2}$$

A média de escritas concluídas foi calculada considerando as cinco execuções realizadas para cada nível de carga. Essa métrica foi utilizada para representar o comportamento médio da arquitetura em cada cenário avaliado. O cálculo da média foi realizado a partir da soma do percentual de escritas concluídas em cada execução, representado por E_i, dividida pelo número total de execuções, conforme apresentado na Equação 3.

$$\text{Média de Escritas Concluídas} = \frac{\sum_{i=1}^{5} E_i}{5} \tag{3}$$

5.1. Cenário sem balanceamento de carga

Como ponto de partida, observa-se que com a taxa de 15 requisições por segundo, a plataforma consegue persistir todos os dados sem perdas. Até 100 RPS, o sistema mantém uma taxa de erro abaixo do 1%. No entanto, no intervalo entre 200 e 500 RPS, ocorre uma redução gradual no percentual de escritas concluídas.

No cenário de 1000 RPS, observa-se uma queda mais acentuada no percentual de escritas concluídas, atingindo valores próximos a 94%, enquanto a taxa de erro ultrapassa 8% em algumas execuções. Esse comportamento indica que a plataforma enfrentou

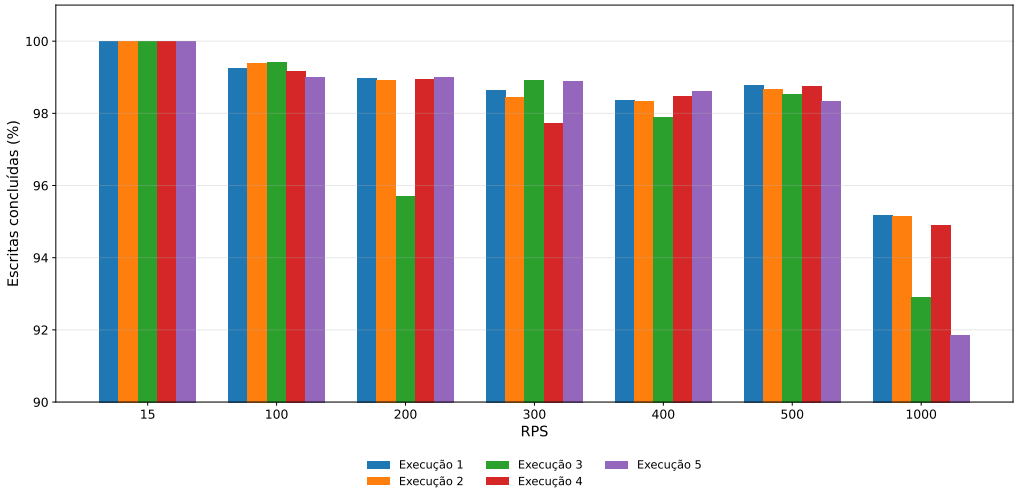

Figura 4. Taxa de escritas concluídas no cenário sem balanceamento de carga

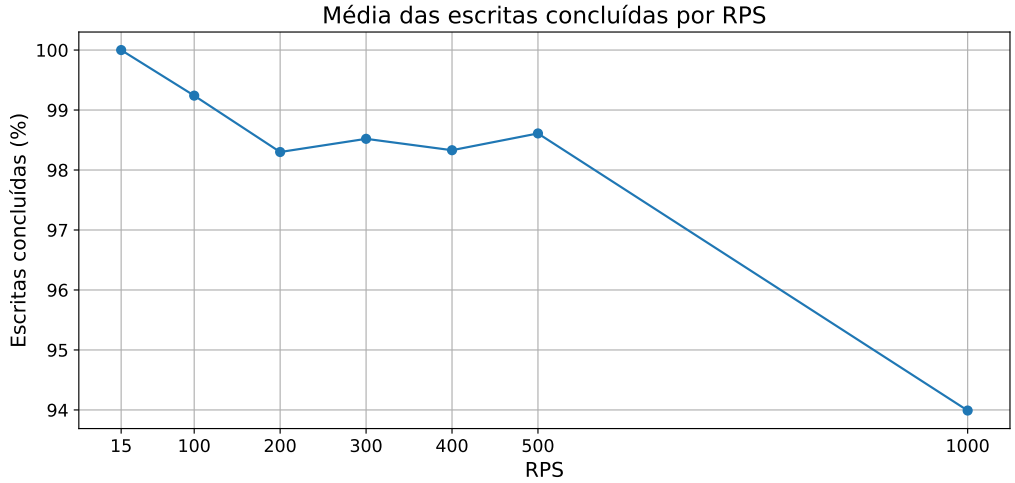

Figura 5. Média das escritas concluídas no cenário sem balanceamento de carga

limitações operacionais nesse nível de carga, comprometendo sua capacidade de processar e persistir requisições de forma eficiente.

Além disso, na Figura 5, a média das escritas concluídas mantém-se relativamente estável no intervalo entre 200 e 500 RPS, com variações entre 98% e 99% de escritas concluídas. Entretanto, ao elevar a carga para 1000 RPS, ocorre uma redução significativa, evidenciando o impacto do aumento da carga na capacidade da plataforma em persistir dados.

5.2. Cenário com balanceamento de carga

A Figura 6 apresenta os resultados do segundo cenário em que os testes foram realizados com balanceamento de carga entre os clientes MQTT. Assim como no primeiro cenário, para a taxa de 15 RPS houve 100% de sucesso nas escritas. No entanto, não foi identificada uma melhora significativa no desempenho em relação ao cenário sem balanceamento, indicando que a adoção de múltiplas instâncias do LoRaWAN IoT Agent não resultou em ganhos expressivos na taxa de escritas concluídas.

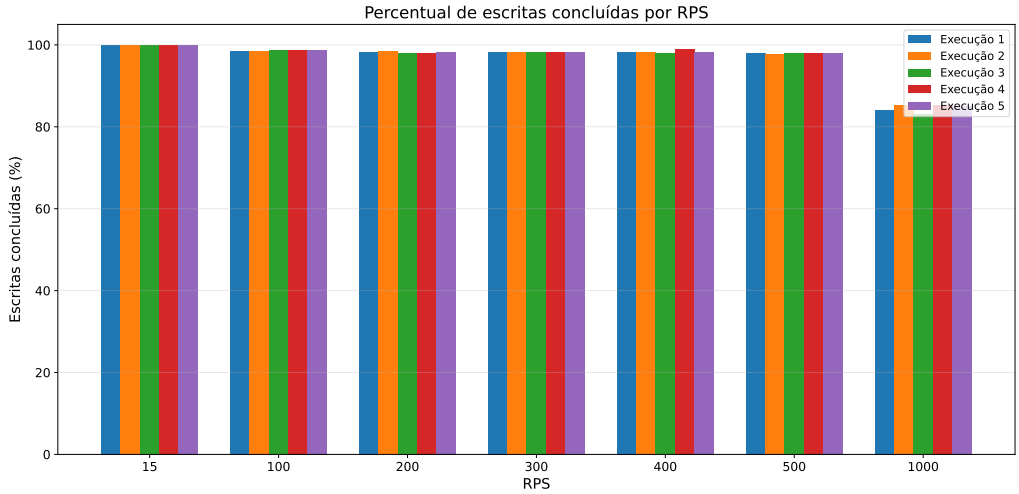

Figura 6. Taxa de escritas concluídas no cenário com balanceamento de carga

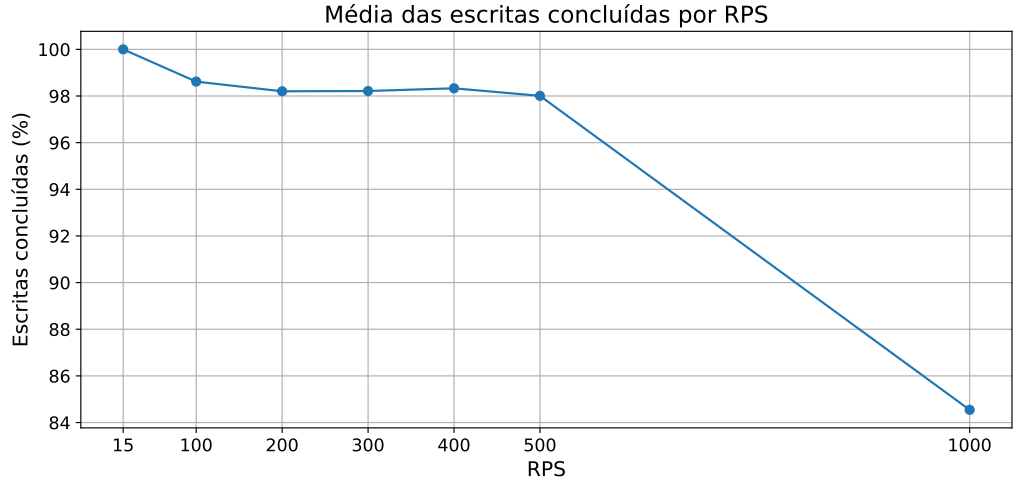

Figura 7. Média das escritas concluídas no cenário com balanceamento de carga

Conforme ilustrado na Figura 7, a média das escritas concluídas no cenário com balanceamento de carga mantém-se mais estável no intervalo entre 200 e 500 RPS. Entretanto, ao se elevar a carga para 1000 RPS, observa-se uma degradação mais acentuada no desempenho em comparação ao cenário sem balanceamento, com médias inferiores a 86% e taxas de erro superiores a 16%. Esses resultados indicam que, apesar de apresentar certa estabilidade em cargas intermediárias, o balanceamento de carga não foi suficiente para sustentar o desempenho em níveis mais elevados de requisição.

A Figura 8 apresenta o consumo de CPU dos principais contêineres da arquitetura FIWARE durante os testes de carga. Observa-se que o *Orion Context Broker* foi um dos componentes com maior utilização de CPU, atingindo aproximadamente 62,2%, indicando que o gerenciamento de contexto e o processamento das entidades representam um dos principais gargalos da arquitetura.

Além disso, os componentes relacionados aos *IoT Agents* também apresentaram elevado consumo computacional. O contêiner *docker-iotagent-lora-1* atingiu cerca de 26%

Figura 8. Dados do Monitoramento com o Prometheus

de utilização média de CPU, evidenciando o impacto do processamento das mensagens MQTT e da comunicação entre dispositivos IoT e o Orion.

6. Conclusões e Trabalhos futuros

Este trabalho avalia a escalabilidade da plataforma FIWARE com o objetivo de verificar sua capacidade de suportar a execução de aplicações de Cidades Inteligentes. Os testes foram realizados utilizando a ferramenta JMeter, além da aplicação do mecanismo MQTT *Shared Subscriptions* para garantir a distribuição de carga entre clientes MQTT.

Estes dois cenários foram fundamentais para a avaliação das estratégias de escalabilidade da plataforma, demonstrando que a escalabilidade horizontal por meio da replicação dos LoRaWAN IoT Agents não se mostrou a abordagem mais eficaz para garantir a persistência dos dados em cenários de testes de carga.

Para taxas menores testadas no trabalho é possível notar que a plataforma realiza a persistência de 100% dos dados. No entanto, ainda existem desafios relacionados à garantia da persistência de dados em níveis mais elevados de carga, especialmente em cenários com requisições superiores a 1000 RPS.

Adicionalmente, sugere-se a análise da escalabilidade vertical de outros componentes da plataforma, a fim de compreender melhor o impacto do aumento de recursos computacionais no desempenho global do sistema.

Referências

Araujo, V., Mitra, K., Saguna, S., and Åhlund, C. (2019). Performance evaluation of fiware: A cloud-based iot platform for smart cities. *Journal of Parallel and Distributed Computing*, 132:250–261.

Chan, V. W. S. (2021). Internet of things and smart cities. *IEEE Communications Magazine*, 59(10):4–6.

de M. Del Esposte, A., Santana, E. F. Z., Kanashiro, L., Costa, F. M., Braghetto, K. R., Lago, N., and Kon, F. (2019). Design and evaluation of a scalable smart city software platform with large-scale simulations. *Future Generation Computer Systems*, 93:427–441.

Domingos, D. C. P., Ismael, M. A. C., Galante, G., Prasniewski, E., Assunção, W. K. G., Oyamada, M. S., de Bona, L. C. E., and de Camargo, E. T. (2024). Scalability evaluation

of the sentilo platform running on containerized resources. *Proceedings of the Brazilian Symposium on Computing Systems Engineering (SBESC)*.

Eclipse Foundation (2024). Eclipse mosquitto. Acessado em: 29 mar. 2026.

Fernández-Añez, V., Fernández-Güell, J. M., and Giffinger, R. (2018). Smart city implementation and discourses: An integrated conceptual model – the case of vienna. *Cities*, 78:4–16.

FIWARE (2024a). Cygnus documentation.

FIWARE (2024b). LoRaWAN IoT Agent documentation.

FIWARE (2024c). Orion Context Broker documentation.

FIWARE Foundation (2026). About us. `https://www.fiware.org/about-us/`.

HiveMQ (2024). Mqtt 5 essentials part 7: Shared subscriptions. Acessado em: 29 mar. 2026.

Martínez, R., Ángel Pastor, J., Álvarez, B., and Iborra, A. (2016). A testbed to evaluate the fiware-based iot platform in the domain of precision agriculture. *Sensors*, 16(11):1979.

MongoDB (2024). MongoDB: The developer data platform.

Monzón, A. (2015). Smart cities concept and challenges: Bases for the assessment of smart city projects. In *Proceedings of the 2015 International Conference on Smart Cities and Green ICT Systems (SMARTGREENS)*, pages 1–11.

Panfilis, G. D. (2021). Fiware integrated iot platform managing product life cycle. Technical report, FIWARE Foundation.

PostgreSQL Global Development Group (2024). PostgreSQL: The World's Most Advanced Open Source Relational Database.

Rizi, M. H. P. and Seno, S. A. H. (2022). A systematic review of technologies and solutions to improve security and privacy protection of citizens in the smart city. *Internet of Things*, 20:100584.

Santana, E. F. Z., Chaves, A. P., Gerosa, M. A., Kon, F., and Milojicic, D. S. (2016). Software platforms for smart cities: Concepts, requirements, challenges, and a unified reference architecture. *ACM Computing Surveys*, 49(3):1–37.

Santana, E. F. Z., Chaves, A. P., Gerosa, M. A., Kon, F., and Milojicic, D. S. (2017). Software platforms for smart cities: Concepts, requirements, challenges, and a unified reference architecture. *ACM Computing Surveys*, 50(6):1–37.

Wenge, R., Zhang, X., Cooper, D., Chao, L., and Hao, S. (2014). Smart city architecture: A technology guide for implementation and design challenges. *China Communications*, 11(3):56–69.

Zanella, A., Bui, N., Castellani, A., Vangelista, L., and Zorzi, M. (2014). Internet of things for smart cities. *IEEE Internet of Things Journal*, 1(1):22–32.