UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Gabriel Aparecido Oliveira Nunes

Mecanismo de Orquestração Dinâmica de Recursos em Cloud Computing para Ambientes IoT

Uberlândia, Brasil

2024

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Gabriel Aparecido Oliveira Nunes

Mecanismo de Orquestração Dinâmica de Recursos em Cloud Computing para Ambientes IoT

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, como parte dos requisitos exigidos para a obtenção título de Bacharel em Sistemas de Informação.

Orientador: Prof. Dr. Diego Nunes Molinos

Universidade Federal de Uberlândia – UFU

Faculdade de Computação

Bacharelado em Sistemas de Informação

Uberlândia, Brasil

2024

Gabriel Aparecido Oliveira Nunes

Mecanismo de Orquestração Dinâmica de Recursos em Cloud Computing para Ambientes IoT

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, como parte dos requisitos exigidos para a obtenção título de Bacharel em Sistemas de Informação.

Trabalho aprovado. Uberlândia, Brasil, 21 de novembro de 2024:

Prof. Dr. Diego Nunes Molinos
FACOM/UFU
Orientador

Prof. Dr. Ivan da Silva Sendin
FACOM/UFU

Prof. Dr. Rodrigo Sanches Miani
FACOM/UFU

Uberlândia, Brasil
2024

Dedico este trabalho aos meus pais, por seu amor incondicional e pelo apoio em cada passo desta caminhada. Sem o incentivo e a confiança de vocês, esta conquista não seria possível .

Agradecimentos

Agradeço primeiramente à minha família, pelo amor, apoio incondicional e incentivo ao longo desta jornada acadêmica. Aos meus pais, por serem minha fonte de inspiração e por acreditarem no meu potencial. Expresso minha profunda gratidão ao meu orientador, Professor Diego Nunes Molinos, pela orientação, paciência e valiosas contribuições que foram essenciais para a realização deste trabalho. Aos meus colegas e amigos, que compartilharam comigo momentos de aprendizado e crescimento, meu sincero agradecimento pelas trocas de conhecimentos e pelo companheirismo. Manifesto também meu reconhecimento à sociedade brasileira, que, por meio de seus impostos, torna possível a existência de universidades públicas federais. Estudar em uma instituição custeada pela contribuição de todos os cidadãos é um privilégio pelo qual sou imensamente grato. Espero, retribuir de alguma forma esse investimento, contribuindo para o desenvolvimento do nosso país. Por fim, agradeço a todos os professores e funcionários da Universidade Federal de Uberlândia, que direta ou indiretamente contribuíram para minha formação acadêmica e pessoal.

Resumo

Em cenários de Internet das Coisas (IoT), os dispositivos, naturalmente espalhados na borda da rede, possuem restrições de capacidade computacional e limitações de memória. Isso os torna inadequados para realizar processamentos intensivos ou armazenar grandes volumes de dados, necessitando, assim, de suporte de infraestrutura mais próxima. Nesse contexto, emerge o conceito *Edge Computing*, que se propõe a deslocar parte do processamento e armazenamento para a borda da rede, mais próxima da fonte dos dados. Assim, em vez de enviar um grande volume de dados para *data centers* centralizados, os dispositivos IoT podem processar esses dados localmente. Esta descentralização oferece vantagens significativas, como por exemplo a redução de latência. Como resultado, dispositivos e aplicações IoT se beneficiam de respostas mais rápidas, garantindo eficiência e eficácia nas operações. Considerando a diversidade e a imprevisibilidade das cargas de trabalho geradas pelos dispositivos de IoT, a ideia de um mecanismo de orquestração dinâmica de recursos de rede para ambientes IoT se apresenta como uma proposta flexível e escalável frente às necessidades desses cenários. Tal proposta não apenas deve ser capaz de se adaptar em tempo real às variações da demanda, mas também garantir a eficiência no uso dos recursos disponíveis da rede. Os experimentos realizados permitiram validar a capacidade do orquestrador em gerir a latência dos dispositivos IoT, demonstrando sua eficiência na identificação de latências elevadas e na aplicação de políticas de QoS. Isso confirma que a abordagem proposta é viável e benéfica para ambientes IoT, promovendo uma gestão eficiente e adaptativa dos recursos de rede.

Palavras-chave: *Edge Computing*, Orquestração, Recursos de Rede, Internet das Coisas.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

IoT	Internet of Things
SOA	Service Oriented Architecture
QoS	Quality of Service
NFV	Network Functions Virtualization
SDN	Software-Defined Networking
VPN	Virtual Private Network
CI	Smart City
DDS	Data Distribution Service
IaaS	Infrastructure as a Service
PaaS	Platform as a Service
SaaS	Software as a Service
TC	Traffic Control

Sumário

1 Introdução

A computação em nuvem tem se firmado como uma ferramenta revolucionária, modificando fundamentalmente os paradigmas tradicionais de armazenamento, processamento e acesso a dados. Paralelamente, a emergência da Internet das Coisas (IoT) trouxe consigo um novo cenário, no qual inúmeros dispositivos, de simples sensores de temperatura domésticos a sofisticados sistemas industriais, estão constantemente gerando e trocando informações em tempo real.

Devido a centralização de diversos recursos na tradicional arquitetura da computação em nuvem, o interesse pela computação de borda, ou *Edge Computing*, aumentou, principalmente devido a sua capacidade de análise e processamento desses dados próximos à fonte, permitindo respostas mais rápidas e eficientes, e reduzindo a sobrecarga em data centers centralizados (CAO et al., 2020).

Conforme citado em Donno, Tange e Dragoni (2019), espera-se que até 2025 mais de 40 bilhões de dispositivos estejam interconectados por meio da IoT, o que representa um imenso desafio de processar e administrar um volume crescente de dados. Quando geridos adequadamente, esses dados têm o potencial de revolucionar áreas como a saúde, com monitoramento em tempo real de pacientes; a mobilidade urbana, por meio da otimização de rotas e gestão de tráfego; e até mesmo a gestão de energia, permitindo cidades mais sustentáveis e eficientes.

A orquestração desempenha um papel central na gestão de sistemas distribuídos, especialmente em ambientes complexos como a computação em nuvem e a IoT. De acordo com (TOMARCHIO; CALCATERRA; MODICA, 2020) a orquestração de recursos na nuvem consiste em atividades como seleção, configuração, implantação, monitoramento e controle em tempo de execução de recursos e aplicativos. O objetivo principal é garantir a entrega contínua e eficiente de serviços, atendendo aos requisitos de QoS tanto para os provedores quanto para os usuários finais. Esse processo automatizado é essencial para lidar com a complexidade e a heterogeneidade das infraestruturas modernas de computação.

Dessa forma, a orquestração pode ser vista como o elemento chave para gerenciar de forma eficiente os recursos em sistemas distribuídos, garantindo que as demandas de um ambiente heterogêneo sejam atendidas de maneira dinâmica e escalável. Este trabalho adota essa perspectiva para explorar soluções de orquestração que otimizem o desempenho e a eficiência de ambientes de computação voltados para IoT.

No contexto de gerência de recursos de rede, sem uma orquestração eficiente, os sistemas podem enfrentar sobrecargas, resultando em latências elevadas e possíveis falhas

(ALAM et al., 2018). Diante disso, o objetivo principal deste trabalho é desenvolver uma solução de orquestração dinâmica que permita a alocação e gestão eficaz de recursos em ambientes de computação em nuvem voltados para IoT.

1.1 Justificativa

A IoT trouxe um aumento exponencial na geração de dados, com dispositivos constantemente trocando informações em tempo real, o que gera demandas específicas para os ambientes computacionais. No entanto, a eficácia dessa troca de informações depende de uma infraestrutura capaz de atender a requisitos como baixa latência, escalabilidade e diversidade de protocolos de comunicação, características inerentes aos ambientes IoT (ALAM et al., 2018)

A computação em nuvem oferece flexibilidade e escalabilidade, permitindo o provisionamento rápido de recursos conforme a demanda (TAURION, 2009). No entanto, a dinâmica e a heterogeneidade dos dispositivos IoT introduzem desafios específicos, como variações na carga de trabalho, requisitos de latência e diversidade de protocolos de comunicação (DONNO; TANGE; DRAGONI, 2019).

A orquestração dinâmica de recursos surge como uma abordagem essencial para garantir a eficiência operacional em ambientes de computação voltados para IoT. Ela possibilita a alocação eficiente de recursos de computação, armazenamento e rede, garantindo que cada componente da infraestrutura esteja dimensionado de acordo com a sua demanda real, evitando desperdícios (ALAM et al., 2018; DIKAIAKOS et al., 2009).

Ambientes IoT estão sujeitos a variações rápidas nas condições de operação de sensores e atuadores, sendo essencial, um mecanismo de orquestração dinâmica para adaptar-se a essas mudanças, reconfigurando automaticamente a alocação de recursos para atender às novas demandas e garantir a continuidade das operações (ALAM et al., 2018). Entende por recursos em ambientes IoT a latência da comunicação, segurança e conformidade e a escalabilidade da solução.

Em suma, o desenvolvimento de um mecanismo de orquestração dinâmica para ambientes de computação em nuvem voltados para cenários de IoT é essencial para superar os desafios específicos desse cenário, proporcionando eficiência operacional, adaptabilidade e otimização de recursos em tempo real. A implementação bem-sucedida desse mecanismo resultará em benefícios tangíveis, como melhor desempenho, menor custo operacional e maior confiabilidade na infraestrutura de IoT baseada em nuvem.

1.2 Objetivos

O objetivo deste trabalho é projetar e validar um mecanismo de orquestração dinâmica de recursos em computação em nuvem para ambientes IoT.

Estabelecem-se como objetivos específicos: (a) analisar e compreender recursos em computação em nuvem, (b) compreender modelos de orquestração para computação em nuvem, (c) projetar e validar um mecanismo de orquestração dinâmica para ambientes IoT, (d) implementar e avaliar o modelo proposto.

1.3 Divisão da Monografia

A divisão do trabalho compreende a fundamentação teórica que engloba toda a conceitualização necessária para compreensão deste trabalho, a metodologia que detalha os métodos de construção da solução proposta, desenvolvimento que irá detalhar aspectos da arquitetura da solução e a codificação, a seção de análise e resultados irá apresentar os testes e resultados obtidos e por fim a seção de conclusão.

2 Revisão Bibliográfica

Este capítulo apresenta os conceitos, paradigmas e tecnologias que sustentam as discussões e soluções propostas, além de explorar os trabalhos já realizados nas áreas de interesse, identificando lacunas e justificando a relevância do presente estudo.

Inicialmente, será discutido o conceito de Computação em Nuvem, destacando seu papel como uma solução amplamente adotada para o provisionamento de recursos escaláveis e sua relevância em ambientes de IoT. Em seguida, será abordada a Computação de Borda, enfatizando como esse paradigma complementa a computação em nuvem ao trazer capacidades de processamento e análise para mais perto da origem dos dados, reduzindo a latência e otimizando recursos.

O capítulo também analisa a Orquestração de Recursos de Rede em Ambientes de Computação em Nuvem, discutindo como essa prática possibilita a coordenação eficiente de recursos distribuídos e heterogêneos, com foco na garantia de QoS. Em seguida, será explorado o Ecossistema de IoT, apresentando suas características, camadas arquiteturais e desafios inerentes à integração com paradigmas computacionais modernos. Por fim, serão apresentados os Trabalhos Relacionados, destacando pesquisas e soluções que abordam temas semelhantes ao deste estudo.

Essa revisão oferece uma visão abrangente das bases teóricas e práticas que sustentam este estudo, permitindo identificar oportunidades de inovação e estabelecer os fundamentos para as contribuições propostas.

2.1 Computação em Nuvem

De acordo com Taurion (2009), após uma era de ascensão tecnológica, miniaturização de componentes eletrônicos e notável avanço do computador pessoal, observou-se um período na evolução caracterizado pela descentralização da computação, fazendo uso do processamento e armazenamento da informações dispersos em cada computador. Ainda que, adquirir um computador não exige investimento financeiro alto, a integração deste com outros computadores ligados em rede, fazendo uso de inúmeros recursos se apresenta como um investimento alto para empresas e corporações.

Dentro deste contexto, buscando uma maior consolidação dos recursos de rede, poder de computação e integração massiva de computadores e servidores a computação em nuvem, ou mais conhecida como *computação em nuvem*, se apresenta como uma solução elegante frente aos desafios de orquestração de recursos, conectividade e disponibilidade de serviços na rede (TAURION, 2009).

A computação em nuvem trata-se de uma infraestrutura na qual os recursos computacionais são disponibilizados como um serviço para seus usuários através de um modelo baseado em consumo (DONNO; TANGE; DRAGONI, 2019). Este modelo baseia-se em abordagens como a Arquitetura Orientada a Serviços (SOA), a virtualização e a computação em larga escala.

SOA trata-se de um *framework* projetado para a integração de processos de negócios e o suporte à infraestrutura de TI por meio de componentes padronizados e seguros, conhecidos como serviços (NIKNEJAD et al., 2020). A SOA representa uma arquitetura abrangente de TI, incentivando a independência, reutilização e interoperabilidade entre sistemas (BIEBERSTEIN, 2006). Sendo uma arquitetura modular, a SOA proporciona flexibilidade na integração e reutilização de serviços, enquanto sua capacidade de encapsular várias aplicações e fontes de dados oferece transparência, permitindo o acesso a um conjunto integrado de recursos de TI, independentemente da tecnologia, linguagens e plataformas existentes (NIKNEJAD et al., 2020).

Com base nessas abordagens, a computação em nuvem assegura o provisionamento de serviços sob demanda aos seus usuários sempre que houver necessidade, criando a percepção de acesso a recursos computacionais ilimitados (YU; LOU; REN, 2012).

Conforme preconizado por Niknejad et al. (2020), há três modelos principais:

- ***Infrastructure as a Service* (IaaS):** o usuário aluga infraestrutura, como servidores, redes, armazenamento e espaço no data center.
- ***Platform as a Service* (PaaS):** além da infraestrutura, o provedor também oferece plataforma de desenvolvimento, dessa forma, os usuários não precisam se preocupar com a infraestrutura subjacente.
- ***Software as a Service* (SaaS):** o modelo permite que o software seja disponibilizado como um serviço através da internet.

Desta forma, ao adotar soluções de computação em nuvem, as empresas não apenas evitam investimentos significativos na criação de suas próprias infraestruturas, mas também desfrutam de escalabilidade, possibilitando ajustar recursos conforme a demanda; flexibilidade, com a capacidade de se adaptar rapidamente a mudanças e inovações; e eficiência de custos, pois pagam apenas pelo que usam. Essas vantagens, aliadas às necessidades da IoT, tornam computação em nuvem uma solução valiosa tanto para pequenas empresas quanto para grandes corporações (DIKAIAKOS et al., 2009).

2.2 Computação de borda - *Edge Computing*

Computação de borda é um paradigma diferente da computação em nuvem tradicional, ela surge como uma resposta às crescentes demandas por processamento de dados mais próximo de onde eles são gerados (SATYANARAYANAN, 2017; SHI et al., 2016).

Enquanto a computação em nuvem concentra o processamento em *data centers*, a computação de borda distribui essa capacidade pela borda da rede, mais próxima dos locais de origem e consumo de dados. desta forma, busca-se unificar os recursos mais próximos aos usuários, seja em termos geográficos ou de rede (CAO et al., 2020).

O crescimento no uso de IoT apresenta desafios como largura de banda, latência e conectividade, que a computação em nuvem não pode abordar sozinha. Com isso, devido as suas características a computação de borda torna-se uma forte aliada na resolução dos desafios enfrentados em ambientes de IoT (DONNO; TANGE; DRAGONI, 2019).

O funcionamento de uma fábrica inteligente é um bom exemplo, onde várias máquinas e sensores estão constantemente gerando dados sobre a qualidade e eficiência. Em vez de enviar todos esses dados para um data center remoto para processamento, a computação de borda permite que essas máquinas processem e analisem os dados localmente, em tempo real, viabilizando a tomada de decisões instantâneas (BUYYA; SRIRAMA, 2019; SHI et al., 2016).

A Figura 1 ilustra um comparativo visual entre as arquiteturas de computação em nuvem e computação de borda, destacando a centralização do processamento no primeiro caso e a descentralização no segundo.

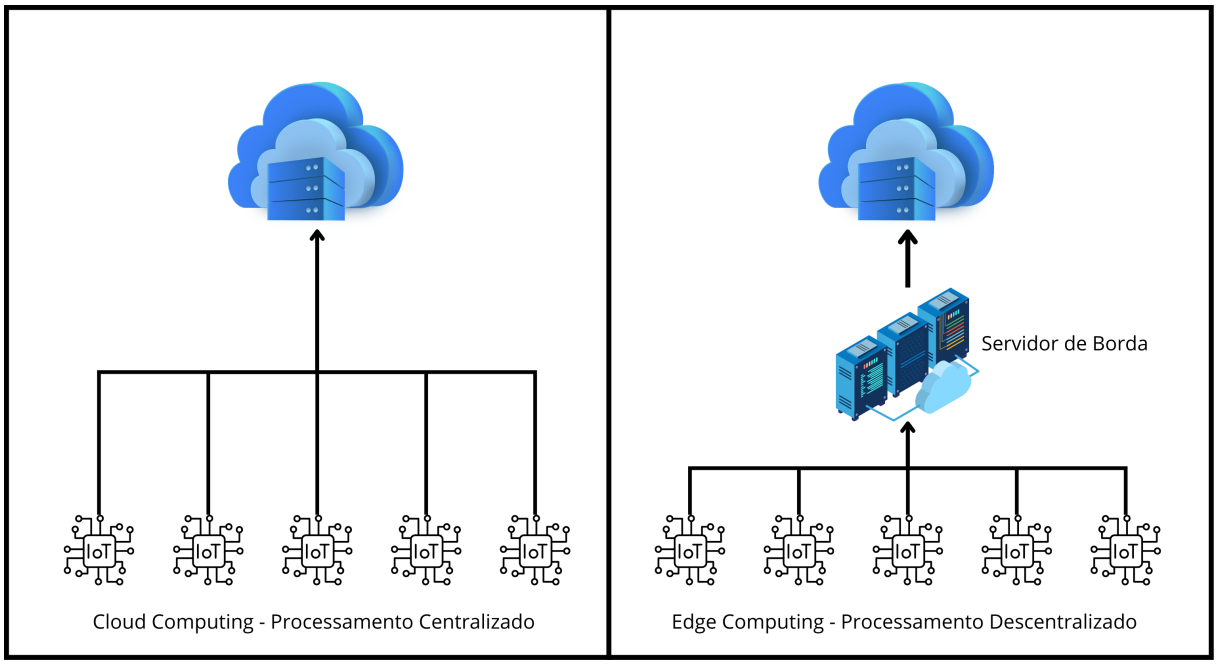

Figura 1 – Diferença entre computação em nuvem e computação de borda

2.3 Orquestração de Recurso de Rede em Ambiente de Computação em Nuvem

A orquestração de recursos na nuvem muitas vezes podem conter operações complexas, como implantação, monitoramento e controle de recursos em tempo de execução. O objetivo geral da orquestração é garantir a disponibilidade das aplicações, atendendo às necessidades de QoS de ambas partes, provedores e consumidores (BARUCHI, 2015).

A orquestração apesar de fornecer inúmeros benefícios não é uma atividade trivial de se realizar, o crescente número de provedores de nuvem heterogêneos é um exemplo da complexidade que se enfrenta na orquestração (SOUZA, 2018; BARUCHI, 2015).

Em computação, um recurso pode ser entendido como qualquer componente físico ou virtual do sistema que pode ser usado para recuperar, processar, armazenar ou transmitir dados. No contexto de IoT, recursos podem referir a dispositivos e sensores, bem como as capacidades desses dispositivos, como processamento de borda, capacidade de armazenamento e conectividade (DING et al., 2018).

Em suma, a orquestração trata a coordenação automática e o gerenciamento de vários sistemas e serviços. Em computação em nuvem, a orquestração refere-se ao processo de organização e coordenação automática de recursos de rede para que seja possível operar conforme o esperado para fornecer serviços específicos ou atender objetivos de negócios predefinidos (BARUCHI, 2015).

2.4 Ecossistema de IoT

Segundo Donno, Tange e Dragoni (2019), pode-se entender Internet das Coisas (IoT) como uma coleção de dispositivos de computação interneconectados através da Internet e destinados a oferecer serviços direcionados a todos os tipos de aplicações. Algumas de suas principais características são a heterogeneidade, ou seja, os dispositivos na IoT podem ser baseados em diferentes redes e/ou plataformas de *hardware*, interagindo com diferentes plataformas e/ou dispositivos por meio de diversas redes e geralmente possuem restrições energéticas e computacionais.

Conforme citado em Donno, Tange e Dragoni (2019), existem vários modelos de arquitetura para IoT, porém, a mais comumente utilizada é baseada em três níveis:

- ***Perception Layer*:** Encarregada de coletar dados do ambiente usando sensores e atuadores.
- ***Network Layer*:** Responsável por transmitir os dados de forma eficiente.

- ***Application Layer*:** Utiliza as informações das duas camadas anteriores para realizar o processamento, que é feito em servidores remotos (computação em nuvem).

2.5 Trabalhos Relacionados

Neste capítulo, são apresentados trabalhos que possuem relação com os objetivos deste projeto. Ora os trabalhos abordam arquitetura e serviços em computação em nuvem, ora abordam a orquestração de recursos em computação em nuvem.

A Tabela 1 destaca os trabalhos analisados sob a óptica do tipo de publicação e o tipo do referido periódico ou evento.

Tabela 1 – Detalhes das Publicações

Título	**Autores**	**Ano**	**Publicação**
Cloud–edge microservices architecture and service orchestration: An integral solution for a real-world deployment experience.	Luis Roda-Sanchez, Celia Garrido-Hidalgo, Fernando Royo, José Luis Maté-Gómez, Teresa Olivares, Antonio Fernández-Caballero	2023	Internet of Things
Orchestration of Microservices for IoT Using Docker and Edge Computing	Muhammad Alam, João Rufino, Joaquim Castro Ferreira	2018	IEEE Communications Magazine
Cidades Inteligentes: Uma arquitetura de Gerenciamento Autonômica no Contexto de IoT	Pablo Tibúrcio, Marcelo Santos, Stênio Fernandes	2017	Anais do Workshop Pré-IETF (WP-IETF)

A abordagem apresentada por Roda-Sanchez et al. (2023) sugere uma solução para a orquestração de serviços e gerenciamento centralizado. O objetivo principal do trabalho é fornecer um sistema escalável, resiliente e ágil, capaz de ser gerenciado remotamente. O fator chave que motivou esse trabalho é a importância de uma gestão eficiente e inteligente de informações, com o foco no processamento de dados o mais próximo possível de sua origem, e a necessidade de uma orquestração dinâmica de serviços com base em recursos disponíveis.

A solução proposta é baseada em microsserviços, projetada para orquestrar serviços e realizar a gestão centralizada de uma rede distribuída que se estende da nuvem até a borda. Seus principais componentes incluem openBalena, BalenaOS, VPN, Firewall, entre outros. Os nós de borda utilizam o BalenaOS e se conectam à plataforma de nuvem através de uma VPN. O openBalena é responsável por gerenciar a implantação de microsserviços nos nós de borda, além de coordenar a comunicação entre os serviços.

A orquestração explanada acontece em um ambiente containerizado, proporcionando flexibilidade e escalabilidade. Além disso, a implantação de novos serviços nos terminais é realizada autonomamente. Os autores concluem que a arquitetura baseada em microsserviços proposta é uma solução viável para implantar aplicativos em nós de borda com recursos limitados. O aplicativo mantém uma carga da CPU e da memória de menos de 50% da capacidade total, mesmo quando usado em dispositivos com recursos limitados. O tempo de implantação também é aceitável, com o aplicativo sendo implantado em menos de 100 segundos se apenas os componentes lógicos forem alterados.

Portanto, embora o artigo não aborde especificamente o tema do trabalho, ele pode contribuir bastante para a solução. A arquitetura supracitada pode servir como base para projetar e implementar o modelo de orquestração dinâmica, além disso o artigo menciona o framework chamado FogFlow que pode servir como referência para desenvolver o modelo.

Em Alam et al. (2018) propõe-se uma arquitetura modular e escalável baseada em virtualização leve. Essa abordagem modular combinada com a orquestração facilitada pelo Docker simplifica o gerenciamento do sistema, oferece suporte à implantação distribuída e promove ambientes dinâmicos. Eles buscam atender aos requisitos de aplicações que operam em IoT, que possuem exigências rigorosas em termos de pontualidade, baixa latência, disponibilidade e confiabilidade. Como saúde inteligente, transporte inteligente e aplicações industriais.

De acordo com Alam et al. (2018), a arquitetura é baseada em três camadas:

- **Camada de borda *(edge layer)*:** Essa camada serve como uma especie de interface mais próxima entre o ambiente físico e os dispositivos conectados. Utilizando Docker e contendo microsserviços especializados conseguem realizar processamento local.
- **Camada intermediária *(middle layer)*:** Essa camada atua como uma ponte entre a camada de borda e a camada de negócio. Age como provedora de pontos de conexão local. Também conhecida como *gateways*, eles são os responsáveis por receberem os dados enviados pela camada de borda, realizar processamento local para reduzir a latência e encaminhar as informações relevantes para a camada de negócio.
- **Camada de negócio *(enterprise layer)*:** Essa camada é a responsável por armazenar e analisar os dados a longo prazo. Dessa forma uma característica dessa camada é o gerenciamento de grandes volumes de dados.

A comunicação entre as camadas e os dispositivos se da a partir de um barramento de mensagens. Cada aplicação possui um canal dedicado, eliminando a comunicação direta

em *publish* e *subscriber*. A arquitetura também possibilita a adição simples de novos dispositivos, para tal feito bastaria somente configurar esses dispositivos para se conectarem ao *gateways*. Segundo os autores, em testes realizados, a arquitetura permitiu a realocação dinâmica de serviços enquanto eles estavam em execução, sem afetar negativamente a disponibilidade ou a operação contínua do serviço. Teve a capacidade de dimensionar dinamicamente o sistema em diferentes camadas. Notou-se também que a introdução do canal dedicado contribuiu para respostas mais rápidas.

No trabalho Tibúrcio, Santos e Fernandes (2017) a arquitetura AutoManIoT é sugerida para o gerenciamento de infraestrutura de rede e dispositivos em ambientes de IoT e cidades inteligentes (CI). Fazendo-se uso dos conceitos de SDN e NFV, a arquitetura incorpora alguns componentes como, controlador SDN, orquestrador, gerente VNF, gerente de infraestrutura virtualizada, ciclo de controle autonômico e elemento de gerência de políticas de rede.

Além disso, Tibúrcio, Santos e Fernandes (2017) fez uso do modelo centrado em dados *DDS (Data Distribution Servce)* que estará a cargo da integração entre aplicações e orquestração de rede. A estrutura organizacional da arquitetura proposta se da por plano de infraestrutura, plano de rede virtual, plano de orquestração NFV, plano de controle, plano de gerência autonômica, plano de conhecimento, controle e gerenciamento global da CI.

A arquitetura AutoManIoT, apresentada em Tibúrcio, Santos e Fernandes (2017), foca em proporcionar uma infraestrutura robusta e flexível para a gerência e virtualização de redes, seu modelo carece de um monitoramento contínuo e dinâmico de métricas de desempenho em tempo real, como latência. Nesse contexto, o presente trabalho complementa essa abordagem ao integrar ferramentas como Prometheus e cAdvisor, possibilitando a coleta e análise de métricas em tempo real, e ao implementar mecanismos de controle de QoS via Ryu. Essa integração oferece um sistema mais adaptativo, capaz de reagir de forma proativa a variações nas condições operacionais, garantindo maior eficiência e confiabilidade na gestão de recursos IoT em ambientes dinâmicos. Por exemplo, enquanto AutoManIoT estrutura a comunicação em planos organizacionais distintos, o modelo proposto incorpora elementos de controle dinâmico que poderiam ser utilizados para otimizar a performance da infraestrutura proposta por AutoManIoT.

Em suma, os trabalhos detalhados nesta seção contribuem para a criação de um modelo de orquestração dinâmica de recursos em computação em nuvem, que se beneficiará das melhores práticas e conceitos apresentados, como, por exemplo, um modelo centrado em dados, que pode ser utilizado para melhorar a comunicação e coordenação entre os serviços na nuvem, ou a integração de SDN e NFV para permitir a virtualização de funções de rede e gestão centralizada, facilitando a adaptação rápida a mudanças.

3 Metodologia

É importante ressaltar que o objetivo do trabalho deve estar bem definido para se tomar qualquer decisão relacionada ao método. Assim, dado que este trabalho possui como máxima projetar e validar um mecanismo de orquestração dinâmica de recursos em computação em nuvem para ambientes IoT, a metodologia deve contemplar as etapas de especificação, desenvolvimento e validação, bem como um etapa de análise e resultados.

Conforme ilustrado na Figura 2, a metodologia proposta segue um fluxo estruturado que inicia com a identificação das necessidades e requisitos do sistema. Em seguida, passa pela especificação e desenvolvimento do mecanismo proposto. Posteriormente, procede-se à validação dos resultados obtidos, culminando na análise e discussão desses resultados para verificar se os objetivos do trabalho foram alcançados.

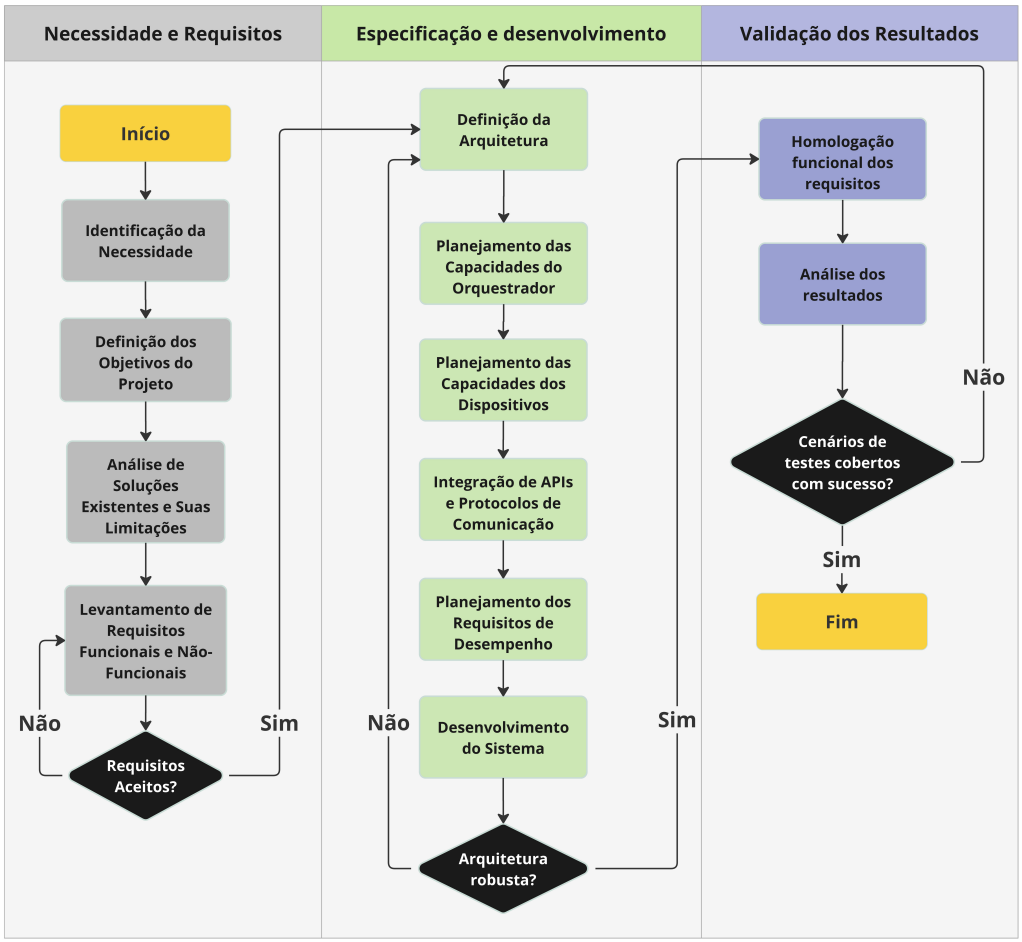

Figura 2 – Fluxograma do método

3.1 Necessidade e Requisitos

No cenário atual de redes de dispositivos IoT, o gerenciamento de recursos e controle de qualidade de serviço são questões de grande relevância, principalmente quando os dispositivos estão em exposição em redes com uma grande quantidade de tráfego e variação de demanda. A necessidade de garantir a latência o mais baixo possível, o controle de banda e o monitoramento de recursos em tempo real são desafios que orientam a solução proposta neste trabalho.

A problemática envolve a necessidade de um orquestrador que, em conjunto com um controlador de rede seja capaz de monitorar e ajustar os parâmetros de rede e dos dispositivos para garantir o melhor desempenho e a qualidade do serviço. O orquestrador deve ser capaz de tomar decisões autônomas com base em métricas de desempenho, viabilizando que a rede seja adaptada e responsiva às demandas.

3.1.1 Levantamento de Requisitos da Aplicação

Diante da problemática apresentada, foi realizado o levantamento de requisitos funcionais e não funcionais da solução baseado na revisão da literatura e trabalhos correlatos. Os requisitos funcionais incluem a coleta de métricas de latência e CPU, a aplicação de políticas de QoS e a comunicação eficiente entre os dispositivos IoT, o orquestrador e o Controlador. Já os requisitos não funcionais englobam aspectos de desempenho e escalabilidade assegurando que o sistema seja robusto.

3.2 Especificação e Desenvolvimento

Com base nos requisitos levantados foi feita a especificação e o planejamento do desenvolvimento da solução. Esta etapa inclui a escolha de ferramentas, interfaces de comunicação e linguagens de programação e componentes que melhor se adéquam ao projeto.

Este processo elenca todos os artefatos do projeto e suas expectativas. No contexto deste trabalho, torna-se importante compreender os elementos que compõem a solução e como irão atuar na arquitetura do projeto:

- ***Orquestrador*:** Aplicação responsável por gerenciar as requisições e provimentos de recursos para dispositivos IoT.
- ***OpenStack*:** OpenStack é uma plataforma de código aberto que atua na gestão de múltiplas infraestruturas virtualizadas. O OpenStack é considerado um Sistema Operacional da Nuvem, por cumprir o mesmo papel em maior escala.

- ***Dispositivos Endpoint*:** Dispositivos IoT responsável por consumir recursos da rede.

Abaixo, detalha-se algumas técnicas e componentes que compõem o mecanismo de orquestração.

- ***Ferramentas*:** O containernet foi escolhido para simular a topologia de rede e dispositivos IoT, enquanto o Ryu foi selecionado como controlador pela sua flexibilidade e suporte a QoS. Para monitoramento de métricas, foram utilizados Prometheus, cAdvisor e Grafana, que coletam dados sobre o uso de recursos dos containers e tornam esses dados acessíveis ao orquestrador, além de visualização de forma gráfica das métricas coletadas.
- ***Interfaces de comunicação*:** A comunicação entre os componentes ocorre via APIs REST, MQTT e consultas ao Prometheus. o Ryu oferece uma API REST que permite ao orquestrador aplicar políticas de QoS, Protocolo OpenFlow 1.3 foi utilizado para comunicação do controlador com os elementos de rede. Os dispositivos IoT simulados se comunicam via MQTT para enviar dados e receber comandos do orquestrador.
- ***Stack de Desenvolvimento*:** O projeto é desenvolvido principalmente em Java com Spring Boot para o orquestrador, utilizando Apache Camel para facilitar a integração de APIs e a execução de tarefas de orquestração. Python é utilizado nos scripts de simulação dos dispositivos IoT e no controlador Ryu.
- ***Componentes*:** Cada dispositivo IoT é simulado como um container Docker gerenciado pelo Containernet, com recursos de CPU e memória configuráveis. O Orquestrador é responsável por coletar métricas de cada dispositivo, analisar esses dados e enviar comandos de ajuste para garantir a qualidade de serviço desejada.

3.3 Validação dos Resultados

Esta etapa trata da demonstração de funcionamento e validação do mecanismo implementado em um ambiente de teste. Esta etapa é dividida em dois momentos, (i) testes funcionais do mecanismo e, (ii) demostrar o funcionamento do mecanismo em um ambiente real de testes, objetivando atender as metas descritas na Tabela 2.

3.4 Análise e Resultados

Como resultados esperados deste trabalho, além da máxima que é projetar e validar um modelo de orquestração dinâmica de recursos em computação em nuvem para

Tabela 2 – Processos a serem avaliados no mecanismo proposto

Ident.	**Métrica**	**Descrição**
MET1	*E1: Validar a comunicação entre dispositivos de borda e orquestrador* *M1: Interface de comunicação dispostivo - orquestrador*	Dispositivos de borda enviam requisições para o orquestrador e capturam respostas
MET2	*E2: Orquestrador atua na gestão de recursos* *M2: Orquestrador consegue de forma dinâmica alocar e liberar recursos para os dispositivos de borda*	Orquestrador consegue a partir de uma requisição atuar na gestão de recursos disponíveis
MET3	*E3: Orquestrador consegue avaliar as requisições* *M3: Orquestrador consegue fe forma dinâmica atuar sobre diferentes requisições*	Orquestrador consegue, a partir de múltiplas requisições, definir prioridade de atuação

ambientes IoT, este trabalho busca aproximar as área de estudo em computação em nuvem e internet das coisas através de um longo aparato conceitual.

4 Desenvolvimento

Nesta seção, será apresentado o desenvolvimento do orquestrador de recursos para dispositivos IoT, que tem como objetivo gerenciar a rede e os dispositivos. A proposta envolve uma solução genérica para um ambiente de IoT, onde diferentes tipos de dispositivos podem ser integrados e gerenciados dinamicamente. O orquestrador é responsável por monitorar métricas como latência e uso de CPU, além de aplicar regras para garantir que a comunicação entre os dispositivos seja realizada de forma eficiente e dentro dos limites de qualidade estabelecidos.

O desenvolvimento foi realizado utilizando um conjunto de tecnologias que, integradas, permitem a criação de um ambiente simulado com dispositivos IoT e controle de rede. Cada ferramenta foi escolhida devido a suas funcionalidades específicas e seu papel na arquitetura geral do sistema. As tecnologias adotadas foram:

- **Apache Camel**(Fundação Apache Software, 2024): Utilizado para criação de rotas de mensagens e facilitar a integração entre sistemas heterogêneos.
- **Containernet**(PEUSTER; KARL; ROSSEM, 2016): Utilizado para emular o ambiente IoT.
- **Ryu**(Ryu SDN Framework, 2024): Controlador SDN (Software Defined Networking), para gestão das necessidades das aplicações.
- **Python e Simulação de Dispositivos IoT**(Python Software Foundation, 2024): Utilizados para simular dispositivos IoT na rede.
- **Docker**(Docker Inc., 2024): Permite a execução de componentes em containers.
- **Prometheus**(Prometheus, 2024), **cAdvisor**(Google Inc., 2024) e **Grafana**(Grafana, 2024): Ferramentas utilizadas para monitorar as métricas.

A estrutura modular deste desenvolvimento possibilita a adaptação e a expansão do orquestrador para diferentes aplicações de IoT. Ao utilizar um ambiente simulado, é possível realizar testes de estresse e aplicar ajustes finos antes de uma eventual implementação no mundo real.

4.1 Arquitetura do sistema

O diagrama da Figura 3 ilustra a arquitetura do sistema desenvolvido para o orquestrador. O sistema é composto por diferentes componentes, cada um desempenhando

um papel específico na coleta de métricas, aplicação de políticas de gerenciamento dos recursos de comunicação entre dispositivos IoT. A arquitetura foi projetada para ser modular, facilitando a integração e monitoramento de diferentes tipos de elementos.

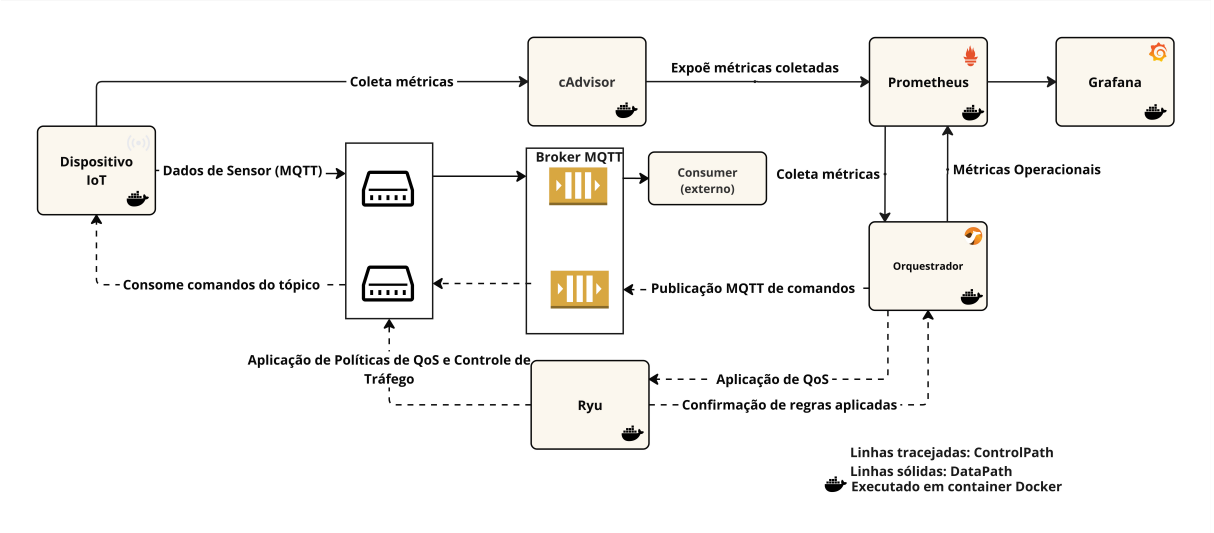

Figura 3 – Arquitetura do sistema de orquestrador IoT

Abaixo serão detalhados os componentes da arquitetura.

- **Dispositivo IoT Simulado:** Este dispositivo simulado executa um script Python que é responsável por enviar dados de sensores via MQTT, representando os dispositivos reais. Ele publica dados no broker, que atua como intermediário para a comunicação entre o dispositivo e os consumidores externos. Além disso, o dispositivo consome comandos do broker, que são enviados pelo orquestrador para ajustes em tempo real, caso sejam necessárias mudanças devido às políticas de QoS.

- **Broker MQTT:** O broker mosquitto atua como intermediário para a troca de mensagens entre o dispositivo IoT, o orquestrador e consumidores externos. Ele recebe os dados de sensores e os repassa para os consumidores para o devido processamento e tomada de decisão. O broker também recebe comandos do orquestrador, que podem ser aplicados aos dispositivos para ajustes de comportamento.

- **Orquestrador Apache Camel:** O orquestrador é responsável por monitorar métricas operacionais como latência e uso de CPU. O monitoramento de latência se dá através de pings periodicos nos dispositivos IoT simulados e o de CPU é coletado através da API REST disponibilizada pelo Prometheus.

- **Controlador Ryu:** Permite o controle programático da rede, aplicando politicas de QoS e controlando o tráfego de dados. Ele recebe instruções do orquestrador para ajustar a largura de banda ou aplicar configurações de QoS. Esse controle é

feito através de um caminho de controle (ControlPath) que é indicado pelas linhas tracejadas no diagrama.

- **cAdvisor:** Coleta métricas dos containers Docker, como uso de CPU e memória, e expõe essas informações para o Prometheus.
- **Prometheus:** Armazena e organiza as métricas, permitindo que o orquestrador consulte esses dados para o monitoramento em tempo real e também envia as métricas para o Grafana.
- **Grafana:** É utilizado para visualizar as métricas operacionais coletadas, como o uso de CPU, memória e trafego de rede. Ele se conecta ao Prometheus para gerar gráficos que facilitam a análise do desempenho do sistema.

4.2 Fluxo de Dados e Controle

- **Data Path (Linhas Sólidas):** As linhas sólidas no diagrama indicam o caminho de dados, onde as mensagens de sensor viajam desde o dispositivo IoT até o orquestrador via Broker MQTT. Também mostra o fluxo de coleta de métricas entre cAdvisor, Prometheus e Grafana.
- **Control Path (Linhas Tracejadas):** As linhas tracejadas representam o caminho de controle. O orquestrador envia comandos de controle para o dispositivo IoT e instruções de QoS para o controlador Ryu. O Ryu, por sua vez, aplica as configurações de QoS e confirma a aplicação ao orquestrador.

4.3 Componentes da Arquitetura

Nesta seção, serão apresentados os principais componentes que compõem a arquitetura proposta para o ambiente IoT simulado.

4.3.1 Containernet

A configuração da rede foi realizada utilizando o Containernet, uma extensão do Mininet que permite a integração com containers Docker. O objetivo desse script é criar uma topologia de rede que simule um ambiente IoT, onde diferentes componentes, como dispositivos que transmitem mensagens via MQTT, um broker, uma aplicação Camel e um servidor Redis para persistência de dados possam se comunicar e interagir por meio de um switch controlado pelo Ryu.

Para iniciar a topologia, a rede é criada e configurada para se conectar a um controlador remoto e em seguida o Ryu é adicionado a rede para gerenciar as decisões

de encaminhamento dentro da rede. O controlador permite que o switch, atuando como núcleo da rede, direcione o trafego e mantenha a conectividade entre os containers. Dessa forma, o Ryu consegue controlar dinamicamente o fluxo de pacotes.

Para conectar o switch a containers Docker externos, são criados pares de interfaces virtuais (veth), esses pares atuam como uma ponte entre o switch e a rede Docker, possibilitando a comunicação entre os containers Docker externos e a rede virtual do Containernet. Para garantir que essa comunicação ocorra com sucesso o script obtém o nome da ponte e a associa a interface chamada docker-s1, estabelecendo uma conexão estável entre o switch e a rede Docker. Além disso, rotas são configuradas em cada container para assegurar que as mensagens sejam corretamente encaminhadas.

Após essa configuração de conectividade, o script escala os containers necessários para o ambiente simulado, incluindo o broker Mosquitto, a aplicação Camel e os dispositivos IoT simulados. Por fim, os endereços IP de cada container são persistidos no Redis, permitindo uma organização centralizada das informações de rede e facilitando o gerenciamento dos dispositivos conectados.

4.3.2 Controlador Ryu

O Ryu é um controlador SDN que permite o gerenciamento centralizado da rede e possibilita a criação de aplicações de controle de rede com configurações dinâmicas e programáveis. No contexto deste projeto, o Ryu é utilizado para aplicar políticas de QoS em dispositivos IoT, ajustando a largura de banda e o encaminhamento de pacotes para otimizar a comunicação e garantir o desempenho da rede.

As políticas de QoS são aplicadas por meio de uma interface REST exposta pelo Ryu. O orquestrador, ao identificar a necessidade de ajustes na rede (por exemplo, quando a latência atinge um limite pré-definido), envia uma requisição HTTP para o controlador, especificando os parâmetros da política a ser aplicada, como limite de largura de banda e prioridade do tráfego.

Por exemplo, para ajustar a largura de banda de um determinado dispositivo, o orquestrador envia uma requisição HTTP ao endpoint do Ryu, conforme o modelo abaixo:

```
POST /qos/queue/{switch_id}
Content-Type: application/json
{
  "port_name": "s1-eth1",
  "type": "linux-htb",
  "max_rate": "5000000"  // em bits por segundo
}
```

Neste exemplo, a requisição cria uma fila de prioridade para o tráfego na porta s1-eth1 do switch especificado, com uma taxa máxima de 5 Mbps. Esse ajuste dinâmico permite que o sistema reaja rapidamente a mudanças nas condições de rede, mantendo a qualidade da comunicação entre os dispositivos IoT.

O monitoramento das métricas dos containers é uma parte essencial para o funcionamento do orquestrador. Para isso, três ferramentas foram utilizadas em conjunto: cAdvisor, Prometheus e Grafana. Essas ferramentas, integradas, possibilitam a coleta, armazenamento e visualização das métricas operacionais dos dispositivos simulados e dos componentes do sistema, como uso de CPU, memória, e tráfego de rede. Abaixo, cada uma dessas ferramentas é detalhada em termos de funcionalidade e papel na arquitetura do sistema.

4.3.3 cAdvisor

O cAdvisor (Container Advisor) é uma ferramenta de monitoramento desenvolvida pelo Google, especificamente projetada para coletar métricas de containers em execução no Docker. No contexto deste projeto, o cAdvisor é responsável por monitorar os containers que simulam os dispositivos IoT e outros componentes do sistema, expondo dados como:

- **Uso de CPU e Memória:** Métricas de utilização de CPU e memória em cada container, permitindo identificar situações de sobrecarga.
- **Tráfego de Rede:** Dados sobre a quantidade de dados transmitidos e recebidos por cada container.

O cAdvisor coleta essas métricas em tempo real e as expõe por meio de uma API, facilitando a integração com o Prometheus para armazenamento e consulta. No projeto, cada container em execução é monitorado pelo cAdvisor, o que permite ao orquestrador obter informações sobre o consumo de recursos e tomar decisões de alocação dinâmica.

4.3.4 Prometheus

O Prometheus é uma ferramenta de monitoramento e armazenamento de séries temporais, amplamente utilizada para coletar e armazenar métricas de sistemas distribuídos. No projeto, o Prometheus é responsável por coletar as métricas expostas pelo cAdvisor e armazená-las em uma base de dados temporal, tornando possível o monitoramento em longo prazo.

A configuração do Prometheus foi realizada para coletar dados dos endpoints do cAdvisor periodicamente, armazenando as métricas para consulta posterior. Além disso, o Prometheus fornece uma linguagem de consulta chamada PromQL (Prometheus Query

Language), que permite ao orquestrador realizar consultas específicas, como a detecção de uso excessivo de CPU e memória. Essas consultas são utilizadas pelo orquestrador para monitorar o estado dos dispositivos e aplicar políticas de QoS quando necessário.

Exemplo de configuração de um job no Prometheus para coletar dados do cAdvisor:

```
scrape_configs:
  - job_name: 'cadvisor'
    static_configs:
      - targets: ['localhost:8080']
```

4.3.5 Grafana

O Grafana é uma plataforma de visualização de dados que permite criar dashboards interativos e gráficos a partir das métricas coletadas pelo Prometheus. No projeto, o Grafana é utilizado para criar dashboards que apresentam o estado atual dos containers, oferecendo uma visão em tempo real dos recursos utilizados por cada dispositivo IoT e pelo sistema em geral.

Os dashboards criados no Grafana incluem gráficos de uso de CPU, memória, e tráfego de rede para cada container, possibilitando uma análise visual das métricas. Essas visualizações ajudam tanto na análise de desempenho quanto na detecção de problemas de forma rápida, facilitando ajustes no sistema.

O fluxo de monitoramento ocorre da seguinte forma:

- **cAdvisor:** coleta as métricas dos containers e as expõe via API.
- **Prometheus:** consulta regularmente o cAdvisor e armazena essas métricas em uma base temporal.
- **Grafana:** utiliza o banco de dados do Prometheus para gerar dashboards interativos, facilitando a análise visual.

4.4 Processamento de Mensagens

O Apache Camel é um framework de integração baseado em Java que simplifica a criação de rotas de mensagens, conectando diferentes sistemas de forma eficiente. No projeto, o Camel foi configurado para trabalhar com Spring Boot, uma escolha que facilita tanto o gerenciamento de dependências quando a inicialização da aplicação.

4.4.1 Configuração da Rota no Camel

No orquestrador, o Camel foi configurado para processar mensagens provenientes de um broker MQTT utilizando o componente Paho, que suporta o protocolo MQTT. Isso permite ao sistema consumir dados de sensores IoT e, após processá-los, enviar comandos a dispositivos conectados.

O trecho de código abaixo ilustra a configuração de uma rota de mensagem no Camel:

```
1    from("paho:iot/sensor/temperature?brokerUrl=tcp
        ://10.0.0.237:1883")
2    .process(new CheckTemperature())
3    .to("paho:iot/sensor/commands/device1?brokerUrl=tcp
        ://10.0.0.237:1883");
```

Essa rota realiza três etapas principais:

1. **Consome mensagens** de um tópico MQTT.
2. **Processa as mensagens recebidas**, aplicando a lógica de negócios.
3. **Publica o resultado do processamento** em outro tópico MQTT.

4.4.2 Explicação Detalhada

- **Origem da Mensagem (`from`)**: O método `from("paho:iot/sensor/temperature?brokerUrl=tcp://10.0.0.237:1883")` define o ponto de entrada da rota. Ele configura o Camel para consumir mensagens publicadas no tópico MQTT `iot/sensor/temperature` utilizando o broker localizado no endereço `tcp://10.0.0.237:1883`.
- **Processador de Mensagens (`process`)**: O método `.process(new CheckTemperature())` permite aplicar a lógica de negócios aos dados recebidos. A classe `CheckTemperature` analisa a mensagem (como o valor de temperatura) e toma decisões, por exemplo, se um comando precisa ser enviado.
- **Destino da Mensagem (`to`)** O método `.to("paho:iot/sensor/commands/device1?brokerUrl=tcp://10.0.0.237:1883"))` publica a mensagem processada no tópico MQTT `iot/sensor/commands/device1`, destinado ao dispositivo IoT. Esse tópico é usado para enviar comandos de controle.

4.4.3 Objetivo e Relevância da Rota

A configuração dessa rota automatiza o fluxo de mensagens entre os dispositivos IoT e o sistema de orquestração. Isso reduz a necessidade de intervenção manual e garante respostas rápidas a eventos capturados pelos sensores.

4.4.4 Exemplo Prático

Imagine um cenário no qual um sensor de temperatura publica um valor acima de 70°C no tópico `iot/sensor/temperature`. A rota Camel processa a mensagem utilizando a classe `CheckTemperature`, que identifica a temperatura elevada e publica um comando no tópico `iot/sensor/commands/device1`, instruindo o dispositivo IoT a ativar um sistema de resfriamento.

5 Experimentos e Resultados

Todos os experimentos foram conduzidos tendo as metas apresentadas na Tabela 2. Neste contexto os experimentos foram conduzidos utilizando 2 cenários, sendo o primeiro responsável por validar as conexões entre os componentes da arquitetura e o segundo cenário responsável por validar a capacidade do orquestrador em atuar sobre os requisitos da comunicação entre os dispositivos.

Para todos os experimentos foi utilizado um computador com as seguintes configurações: (a) **Processador:** Intel(R) Core(TM) i7-8565U CPU @ 1.80GHz - x86_64, (b) **Memória Ram:** 8Gb e (c) **Sistema Operacional:** Linux (Ubuntu 24.04.1 lTS).

5.1 Comunicação dos Componentes da Arquitetura

Conforme definido na Meta 1 da Tabela 2, os testes nesta etapa foram responsáveis por validar a comunicação entre dos dispositivos IoT de borda com o orquestrador. Neste contexto torna-se importante reforçar o fluxo de comunicação previsto na Figura 3.

5.1.1 Dispositos IoT - cAdvisor - Prometheus - Grafana

Uma vez criado todos os dispositivos IoT e instânciado todos os dispositos através de conteineres, os componentes de coleta e exposição de métricas são instânciados. A Figura 4 ilustra a inicialização da ferramenta Prometheus que recebe as métricas expostas pelo cAdvisor.

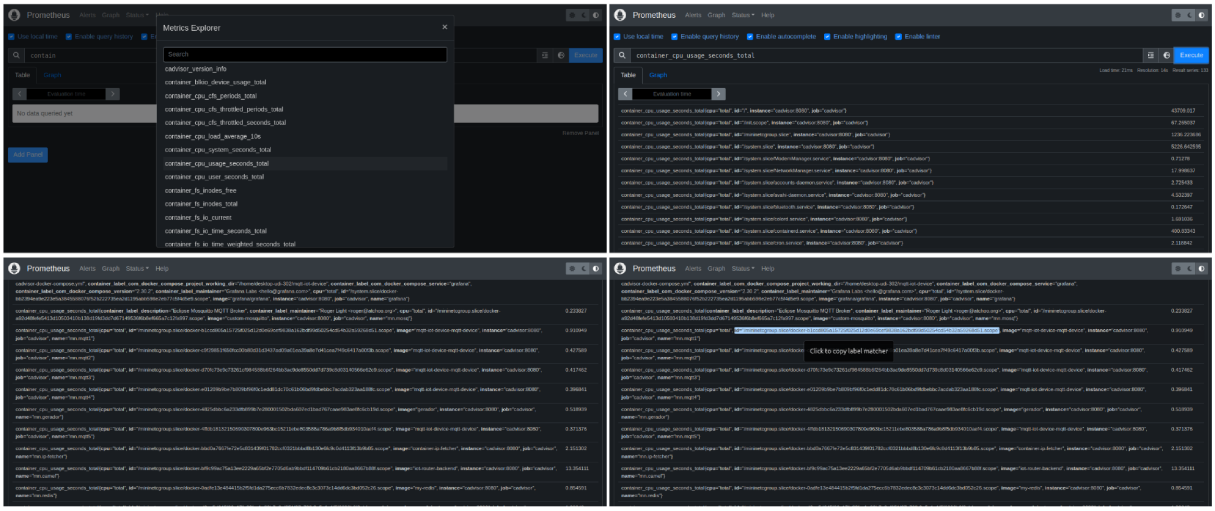

Figura 4 – Inciando Prometheus

Uma vez os dispositivos IoT instânciados e inicializados, as métricas de latência são expostas através do cAdvisor e chegam até o componente Prometheus, foi utilizado o componente Grafana para obter uma melhor vizualização da exposição de métricas dos componentes IoT. A Figura 5 ilustra os dispositivos conectados no Prometheus e as métricas expostas pelos mesmos.

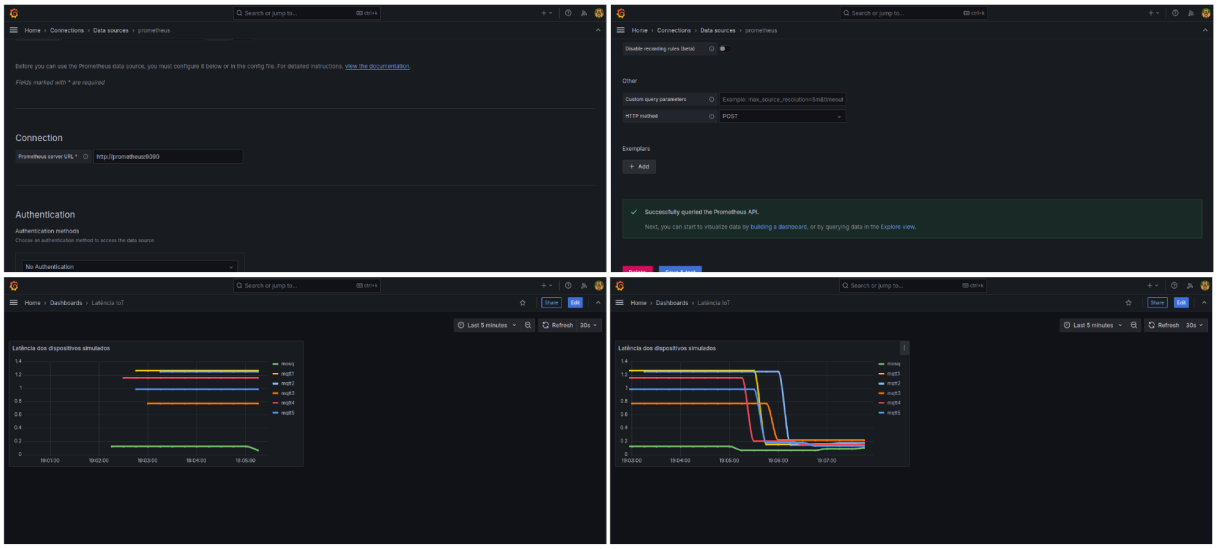

Figura 5 – Inciando Grafana

5.1.2 Orquestrador

Uma vez que os dispositivos encontram-se instanciados, automaticamente eles começam a expor suas métricas para o orquestrador. Nos testes realizados utilizamos apenas a métrica de latência de comunicação, contudo, podem ser adicionadas novas métricas, tais como: métricas de CPU e memória.

Inicialmente as métricas são coletadas pelo componente cAdvisior e expostas para o componente Prometheus que possui uma interface direta com o orquestrador. As Figuras 6 e 7 ilustra a comunicação com o Prometheus para orquestramento das políticas de QoS, neste contexto apenas a latência.

5.1.3 Broker

O broker atua como intermediário para a troca de mensagens entre os dispositivos IoT, o orquestrador e consumidores externos. Ele recebe os dados de sensores e os repassa para os consumidores para o devido processamento e tomada de decisão. O broker também recebe comandos do orquestrador, que podem ser aplicados aos dispositivos para ajustes de comportamento. A Figura 8 ilustra a troca de mensagens entre os dispositos e os componentes da arquitetura.

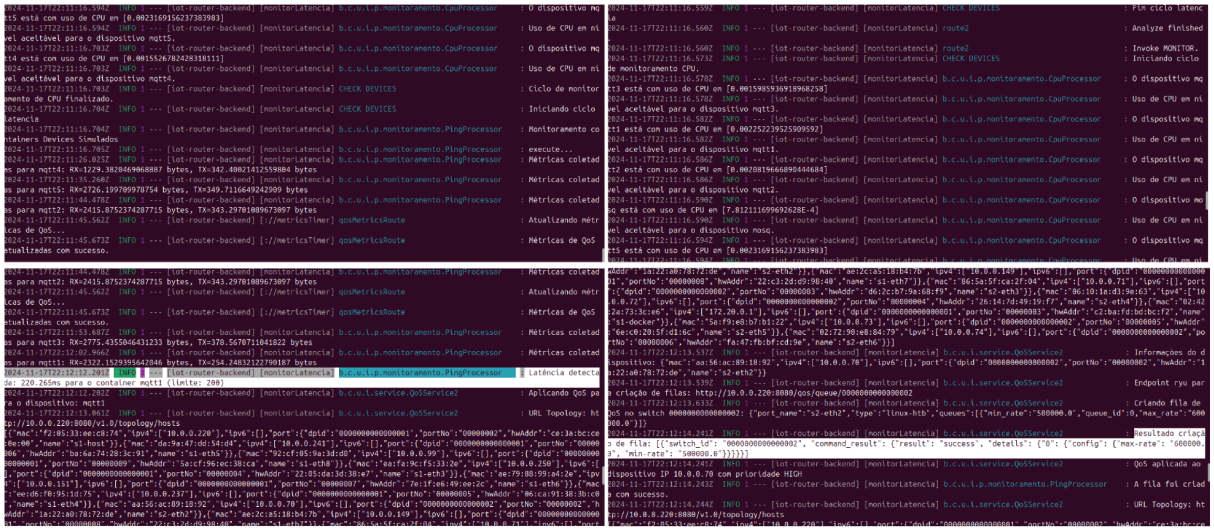

Figura 6 – Inciando Orquestrador

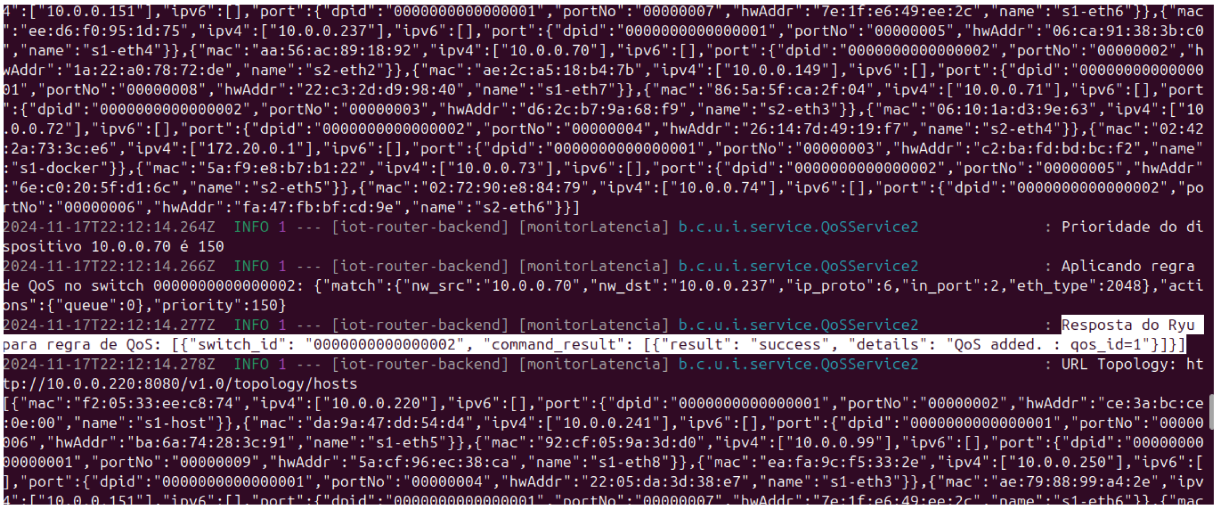

Figura 7 – Inciando Orquestrador

5.2 Controle de Latência

Visando atender as metas 2 e 3 previstas no método, esse cenário busca simular condições de latência elevadas em dispositivos IoT para validar a eficácia do orquestrador em monitorar e mitigar condições adversas na rede. O foco principal está em monitorar dispositivos com latência superior de 200 milissegundos.

As simulações foram realizadas com o uso do TC, uma ferramenta nativa do Linux. O comando foi executado dentro dos containers Docker que representam dispositivos IoT simulados.

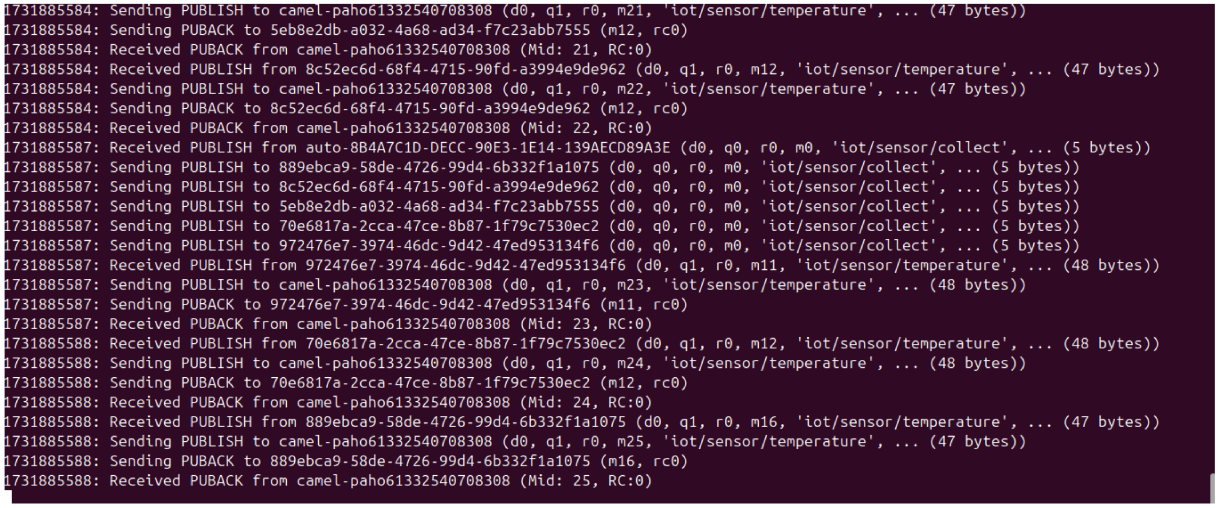

Figura 8 – Mensagens Broker

5.2.1 Experimento - Controle de Latência com 5 Dispositivos

Foram realizados testes com cinco dispositivos simulados denomiados mqtt1, mqtt2, mqtt3, mqtt4 e mqtt5. Todos os dispositivos foram configurados para utilizar 25% de uso de um núcleo de CPU e 256 MB de memória. Entre eles, mqtt1 e mqtt2 foram classificados como dispositivos de alta prioridade, enquanto mqtt3, mqtt4 e mqtt5 foram considerados de baixa prioridade. A Tabela 3 ilustra a configuração inicial dos dispositivos e a latência adicional simulada.

Tabela 3 – Detalhes da Simulação

Dispositivo	**Latência Iniciada (ms)**	**Latência Adicionada (ms)**	**Duração (s)**
mqtt1	0.20	212	60
mqtt2	0.20	208	60
mqtt3	0.20	102	60
mqtt4	0.20	38	60
mqtt5	0.20	21	60

O objetivo desse teste é avaliar a capacidade do orquestrador de detectar e responder adequadamente situações em que um dispositivo apresenta latência acima do limite permitido. Especificamente, a intenção é verificar como o orquestrador identifica essa condição, aplica ajustes nas configurações de rede e, por fim, garante que os dispositivos de alta prioridade mantenham o desempenho dentro dos parâmetros estabelecidos.

Na Figura 9 é apresentado o estado dos dispositivos antes que sofram manipulações.

Figura 9 – Cenário Inicial de Experimento

Após a execução do TC para adicionar o atraso especificado na Tabela 3 podemos observar na Figura 10 um aumento de latência por cerca de quatro minutos, entre 20:50 e 20:54 e além disso, após esse período a latência voltou ao seu estado inicial.

Figura 10 – Ambiente com latência simulada através do TC

Na Figura 11 pode-se observar que no mesmo horário que houve o pico de latência no dispositivo mqtt1 o orquestrador foi capaz de observar essa variância e aplicar regras de controle de latência conforme demonstra as Figuras 12, 13 e 14.

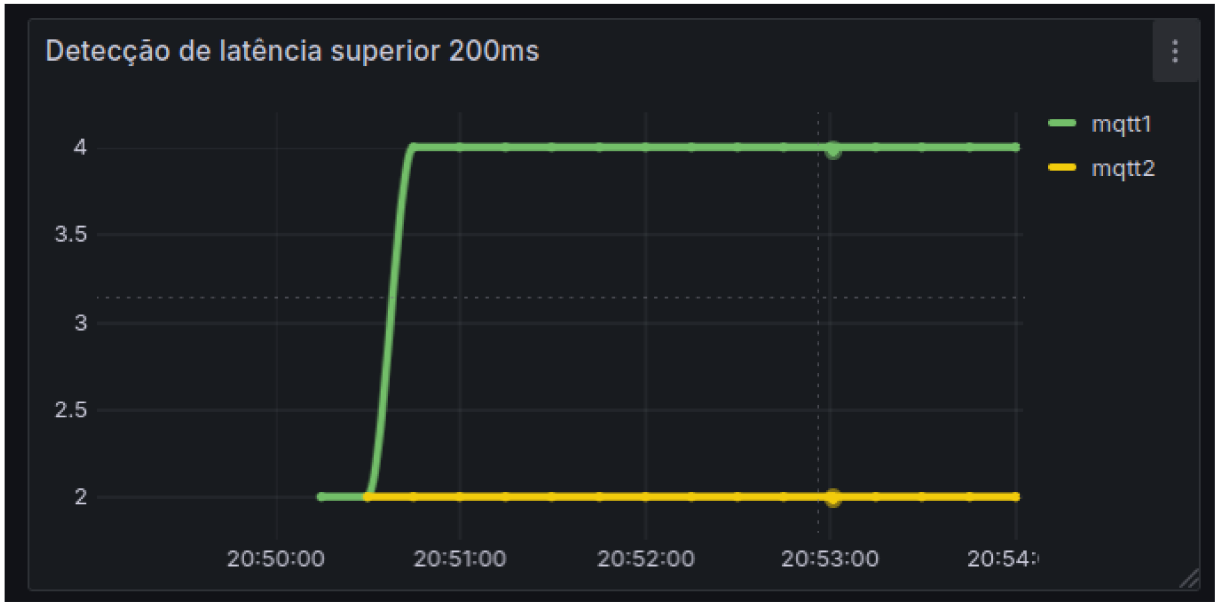

Figura 11 – Orquestrador identificou latência superior 200ms

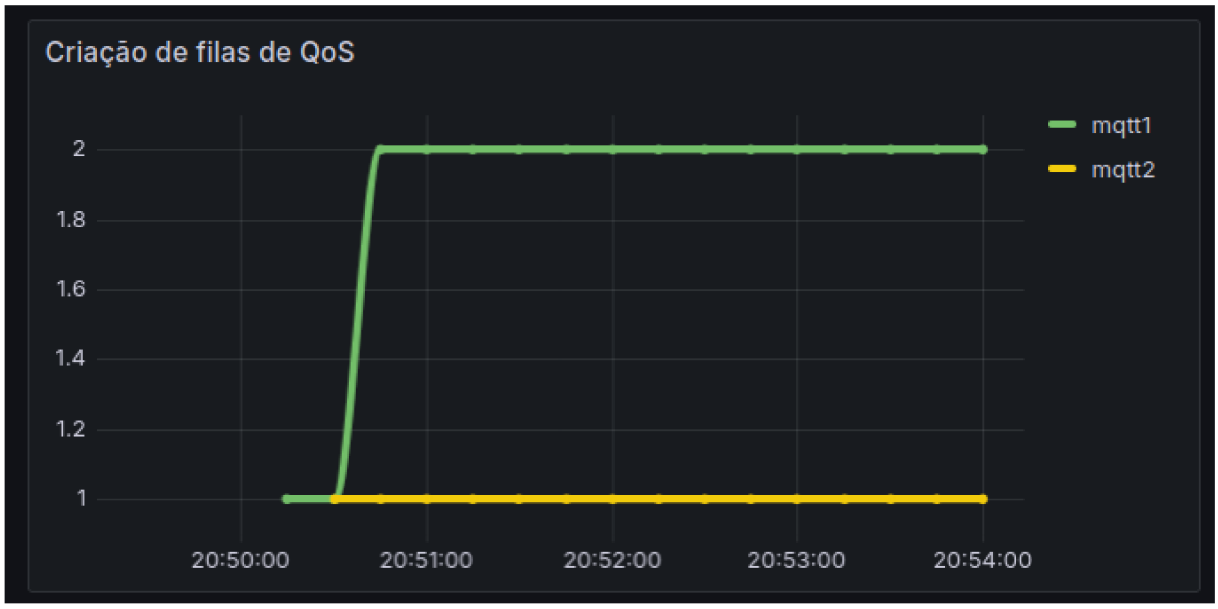

Figura 12 – Criação de filas de QoS para dispositivos mqtt1 e mqtt2 pelo orquestrador

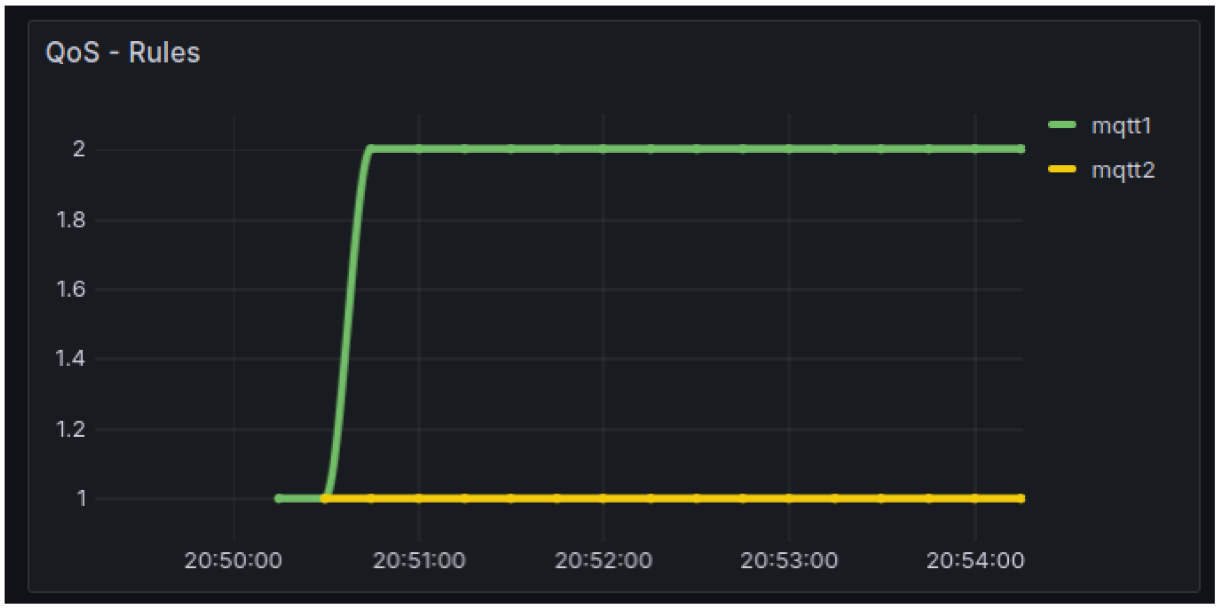

Figura 13 – Associação de fluxos aos parâmetros de filas QoS para os dispositivos mqtt1 e mqtt2

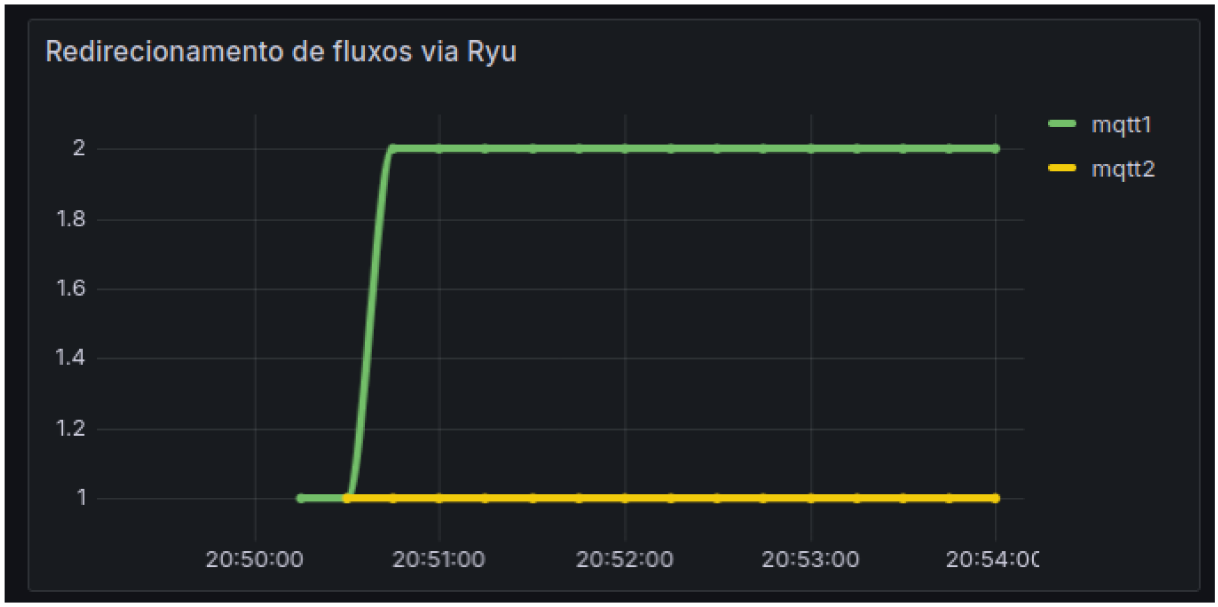

Figura 14 – Redirecionamento de tráfego entre o dispositivo e o broker

6 Considerações Finais

Este trabalho apresentou o desenvolvimento e a validação de um mecanismo de orquestração dinâmica de recursos em ambientes de computação em nuvem voltados para IoT. A proposta destacou-se por sua capacidade de monitorar métricas em tempo real, de acordo com o testes a latência de comunicação dos dispositivos IoT, e ajustar recursos dinamicamente, garantindo maior eficiência na utilização de infraestrutura e mantendo a qualidade de serviço exigida pelos dispositivos IoT.

Os experimentos realizados demonstraram a eficácia do sistema na detecção e mitigação de condições adversas, no caso dos experimetnos a alta latência, além de sua escalabilidade e flexibilidade para atender diferentes tipos de cenários. O uso de tecnologias como Containernet, Ryu, Prometheus e Grafana proporcionou uma base sólida para a simulação e validação do mecanismos, bem como ofertou uma arquitetura escalável e flexível para receber novos parâmetros.

Este trabalho contribui para a literatura ao unir conceitos de computação em nuvem, edge computing e IoT, apresentando um modelo prático e testável que pode ser replicado ou expandido por outros pesquisadores e profissionais da área.

6.1 Trabalhos Futuros

Dentre as possibilidades de desdobramento deste trabalhos estão: (a) Utilização de Inteligência Artificial, ou seja, incorporar técnicas de aprendizado de máquina para prever padrões de uso e realizar ajustes proativos na alocação de recursos; (b) Segurança e Privacidade no que tange a invetigação de mecanismos para fortalecer a segurança do mecanismo, garantindo proteção contra ataques cibernéticos, além de proteger a privacidade dos dados transmitidos; (c) expandir as métricas monitoradas, como por exemplo taxa de utilização de CPU e memória e, (d) Adequar o mecanismos para testes reais.

Referências

ALAM, M.; RUFINO, J.; FERREIRA, J.; AHMED, S. H.; SHAH, N.; CHEN, Y. Orchestration of microservices for iot using docker and edge computing. **IEEE Communications Magazine**, IEEE, v. 56, n. 9, p. 118–123, 2018. Citado 2 vezes nas páginas 12 e 19.

BARUCHI, A. **Orquestração de migração massiva de máquinas virtuais baseada em análise cíclica para ambientes de computação na nuvem.** Tese (Doutorado) — Universidade de São Paulo, 2015. Citado na página 17.

BIEBERSTEIN, N. **Service-oriented architecture compass: business value, planning, and enterprise roadmap**. USA: FT Press, 2006. Citado na página 15.

BUYYA, R.; SRIRAMA, S. N. **Fog and edge computing: principles and paradigms**. John Wiley & Sons, 2019. Disponível em: <https://books.google.com.br/books?id=v4B_DwAAQBAJ>. Citado na página 16.

CAO, K.; LIU, Y.; MENG, G.; SUN, Q. An overview on edge computing research. **IEEE Access**, v. 8, p. 85714–85728, 2020. Citado 2 vezes nas páginas 11 e 16.

DIKAIAKOS, M. D.; KATSAROS, D.; MEHRA, P.; PALLIS, G.; VAKALI, A. Cloud computing: Distributed internet computing for it and scientific research. **IEEE Internet computing**, IEEE, v. 13, n. 5, p. 10–13, 2009. Citado 2 vezes nas páginas 12 e 15.

DING, Z.; OTA, K.; LIU, Y.; ZHANG, N.; ZHAO, M.; SONG, H.; LIU, A.; ZHIPING, C. Orchestrating data as a services-based computing and communication model for information-centric internet of things. **IEEE Access**, IEEE, v. 6, p. 38900–38920, 2018. Citado na página 17.

Docker Inc. **Docker: Build, Ship, and Run Any App, Anywhere**. 2024. Disponível em: <https://www.docker.com/>. Acesso em: 10 nov. 2023. Citado na página 25.

DONNO, M. D.; TANGE, K.; DRAGONI, N. Foundations and evolution of modern computing paradigms: Cloud, iot, edge, and fog. **IEEE Access**, v. 7, p. 150936–150948, 2019. Citado 5 vezes nas páginas 11, 12, 15, 16 e 17.

Fundação Apache Software. **Apache Camel: Um framework de integração baseado em padrões corporativos**. 2024. Disponível em: <https://camel.apache.org/>. Acesso em: 17 nov. 2024. Citado na página 25.

Google Inc. **cAdvisor (Container Advisor)**. 2024. Disponível em: <https://github.com/google/cadvisor>. Acesso em: 17 nov. 2024. Citado na página 25.

Grafana. **Grafana: The open observability platform**. 2024. Disponível em: <https://grafana.com/>. Acesso em: 17 nov. 2024. Citado na página 25.

NIKNEJAD, N.; ISMAIL, W.; GHANI, I.; NAZARI, B.; BAHARI, M. et al. Understanding service-oriented architecture (soa): A systematic literature review and directions for further investigation. **Information Systems**, Elsevier, v. 91, p. 101491, 2020. Citado na página 15.

PEUSTER, M.; KARL, H.; ROSSEM, S. van. Medicine: Rapid prototyping of production-ready network services in multi-pop environments. In: **2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)**. Palo Alto, CA, USA: IEEE, 2016. p. 148–153. Citado na página 25.

Prometheus. **Prometheus: Monitoring system & time series database**. 2024. Disponível em: <https://prometheus.io/>. Acesso em: 17 nov. 2024. Citado na página 25.

Python Software Foundation. **Python Programming Language**. 2024. Disponível em: <https://www.python.org/>. Acesso em: 17 nov. 2024. Citado na página 25.

RODA-SANCHEZ, L.; GARRIDO-HIDALGO, C.; ROYO, F.; MATÉ-GÓMEZ, J. L.; OLIVARES, T.; FERNÁNDEZ-CABALLERO, A. Cloud–edge microservices architecture and service orchestration: An integral solution for a real-world deployment experience. **Internet of Things**, Elsevier, v. 22, p. 100777, 2023. Citado na página 18.

Ryu SDN Framework. **Ryu SDN Framework**. 2024. Disponível em: <https://ryu-sdn.org/>. Acesso em: 17 nov. 2024. Citado na página 25.

SATYANARAYANAN, M. The emergence of edge computing. **Computer**, IEEE, v. 50, n. 1, p. 30–39, 2017. Citado na página 16.

SHI, W.; CAO, J.; ZHANG, Q.; LI, Y.; XU, L. Edge computing: Vision and challenges. **IEEE internet of things journal**, Ieee, v. 3, n. 5, p. 637–646, 2016. Citado na página 16.

SOUZA, G. D. Proposta de nuvem privada multicâmpus para o ifsc usando orquestração de contêineres. 2018. Citado na página 17.

TAURION, C. **Cloud computing-computação em nuvem**. Rio de Janeiro, Brasil: Brasport, 2009. Citado 2 vezes nas páginas 12 e 14.

TIBúRCIO, P.; SANTOS, M.; FERNANDES, S. Cidades inteligentes: Uma arquitetura de gerenciamento autonômica no contexto de iot. In: **Anais do IV Workshop Pré-IETF**. Porto Alegre, RS, Brasil: SBC, 2017. ISSN 2595-6388. Disponível em: <https://sol.sbc.org.br/index.php/wpietf/article/view/3607>. Citado na página 20.

TOMARCHIO, O.; CALCATERRA, D.; MODICA, G. D. Cloud resource orchestration in the multi-cloud landscape: a systematic review of existing frameworks. **Journal of Cloud Computing**, Springer, v. 9, n. 1, p. 49, 2020. Citado na página 11.

YU, S.; LOU, W.; REN, K. Data security in cloud computing. **Morgan Kaufmann/Elsevier, Book section**, Citeseer, v. 15, p. 389–410, 2012. Citado na página 15.

ANEXO A – Artefatos

Encontram-se disponíveis no Github os códigos fontes de toda a solução desenvolvida neste trabalho.

Branch: release/pre-production: <https://github.com/nunesogabriel/mqtt-iot-device>

Branch: release/pre-production: <https://github.com/nunesogabriel/iot-router-backend>