UNIVERSIDADE FEDERAL FLUMINENSE

HEDER DORNELES SOARES

UMA ARQUITETURA ORIENTADA A SERVIÇOS DE NUVEM PARA APOIAR TESTES EXPERIMENTAIS DE SISTEMAS UBÍQUOS

NITERÓI

2021

HEDER DORNELES SOARES

UMA ARQUITETURA ORIENTADA A SERVIÇOS DE NUVEM PARA APOIAR TESTES EXPERIMENTAIS DE SISTEMAS UBÍQUOS

Tese de Doutorado apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do Grau de Doutor em Computação. Área de concentração: CIÊNCIA DA COMPUTAÇÃO

Orientador:
JOSÉ VITERBO FILHO

NITERÓI

2021

Ficha catalográfica automática - SDC/BEE
Gerada com informações fornecidas pelo autor

S676a Soares, Heder Dorneles
Uma Arquitetura Orientada a Serviços de Nuvem Para Apoiar Testes Experimentais de Sistemas Ubíquos / Heder Dorneles Soares ; José Viterbo Filho, orientador. Niterói, 2021.
86 f.

Tese (doutorado)-Universidade Federal Fluminense, Niterói, 2021.

DOI: http://dx.doi.org/10.22409/PGC.2021.d.00926404199

1. Computação Ubíqua. 2. Computação em Nuvem. 3. Internet das Coisas. 4. Produção intelectual. I. Viterbo Filho, José, orientador. II. Universidade Federal Fluminense. Instituto de Computação. III. Título.

CDD -

Bibliotecário responsável: Debora do Nascimento - CRB7/6368

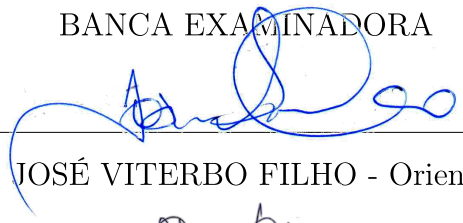
HEDER DORNELES SOARES

UMA ARQUITETURA ORIENTADA A SERVIÇOS DE NUVEM PARA APOIAR TESTES EXPERIMENTAIS DE SISTEMAS UBÍQUOS

Tese de Doutorado apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do Grau de Doutor em Computação. Área de concentração: CIÊNCIA DA COMPUTAÇÃO

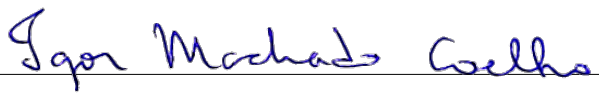
Aprovada 15 em DEZEMBRO de 2021.

BANCA EXAMINADORA



Prof. Dr. JOSÉ VITERBO FILHO - Orientador, UFF

Prof. Dr. FLÁVIA BERNADINI, UFF



Prof. Dr. IGOR MACHADO COELHO, UFF

Prof. Dr. MARKUS ENDLER, PUC-RIO

Dr. LUAN TEYLO GOUVEIA LIMA, Inria Bordeaux

Niterói

2021

Dedico este trabalho aos meus pais Tim e Lia.

Agradecimentos

Agradeço primeiramente aos meus pais Euripedes Dorneles Costa (Tim) e Maria do Carmo Soares (Lia), pelo eterno orgulho, apoio, compreensão, ajuda, e, em especial, por todo carinho ao longo deste percurso. A minha irmã Ludmila Dorneles Soares, pelo carinho, compreensão. E mais recentemente meus sobrinhos Arthur Dorneles Cardoso e Yasmin Dorneles Cardoso.

Ao meu orientador, José Viterbo Filho, não somente pela orientação ou pelo suporte, mas também pelos ensinamentos e a dedicação ao meio acadêmico. E que sem o apoio esse trabalho não seria possível.

Aos amigos da OptHouse: Igor Machado, João Thompson, Marcos Guerine, Matheus Haddad, Pablo Munhoz, Uéverton Souza, Edcarllos Santos, Paulo Rogério, Jânio Carlos, Gelson Schneider e Alan Diêgo. Agradeço não somente pelo incrível acolhimento e aprendizado que adquiri durante todos esses anos, mas também pela convivência e pelos momentos de descontração.

Aos muitos amigos que fiz durante essa etapa: Eduardo Vera, Diego Brandão, Pedro González, Rafael Burlamaqui, Renatha Capua, Marques, Mariana, Sávio, Ítalo, Marcos Raylan e Leonardo. Aos amigos de laboratório: Edhelmira Lima, Pantoja e Leandro. Se me senti bem acolhido nesse lugar com certeza foi por causa de cada um de vocês.

Aos demais professores e funcionários do IC/UFF que direto ou indiretamente contribuíram para tornar possível a concretização deste trabalho.

À CAPES pelo apoio financeiro, sem o qual seria difícil o andamento dos estudos.

Muito obrigado a todos!

Resumo

Sistemas ubíquos estão cada vez mais disponíveis nos mais diversos ambientes, devido a grande quantidade de dispositivos IoT (Internet of Things) que já fazem parte do nosso cotidiano. Porém ainda encontramos alguns desafios técnicos para implementar esses sistemas, como a questão de conectividade de rede, limitações de hardware, interoperabilidade e um importante ponto que é a disponibilidade ou necessidade um de espaço pervasivo para testar novas aplicações e ideias. Usualmente para resolver alguns desses desafios se faz necessário a utilização de diversas camadas de software, o que aumenta a complexidade das soluções e dificulta o processo de desenvolvimento, testes e experimentação. Nos últimos anos tem sido explorado a utilização de tecnologias de nuvem para fornecer uma abstração e uma interface mais simples de acesso a sistemas de testes e experimentação científica. Neste trabalho vamos explorar a utilização de Sensor as a Service (SenaaS) para auxiliar a tarefa de compartilhamento e tratamento dos dados de sensores, e, em conjunto, oferecer um arcabouço para criação de ambientes de testes para esses sistemas, onde o desenvolvedor possa encontrar uma coleção de sensores/atuadores disponíveis para suprir sua necessidade de dados para testar sua aplicação. Além disso, de modo a otimizar o processamento das cargas de trabalho para a arquitetura proposta, dois modelos matemáticos distintos foram desenvolvidos. Um para a alocação mais eficiente de containers e outra para minimização de tempo na implantação de uma tarefa, e a partir da modelagem apresentamos uma simulação dessas atividades.

Palavras-chave: Computação em nuvem, *Sensor as a Service*, computação ubíqua.

Abstract

Pervasive systems are increasingly available in the most diverse environments due to many IoT (Internet of Things) devices that are already part of our daily lives. However, we still face some technical challenges to implementing these systems, such as the issue of network connectivity, hardware limitations, interoperability, and an important point: the availability or need for a pervasive space to test new applications and ideas. Usually, to solve some of these challenges, it is necessary to use several software layers, which increases the complexity of the solutions and makes the development, testing, and experimentation process difficult. In recent years, cloud technologies have been explored to provide a simpler abstraction and interface for accessing test systems and scientific experimentation. In this work, we will explore the use of Sensor as a Service (SenaaS) to assist the task of sharing and processing sensor data, and, together, offer a framework for creating test environments for these systems, where the developer can find a collection of sensors/actuators available to supply your data needs to test your application. Furthermore, two distinct mathematical models were developed to optimize the processing of workloads for the proposed architecture; one for the most efficient allocation of containers and the other for minimizing time in implementing a task, and based on the modeling, we present a simulation of these activities.

Keywords: Cloud Computing, *Sensor as a Service*, Ubiquitous Computing.

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

AAL Assisted Ambient Living

ACL Access Control List

AmI Ambient Intelligence

AWS Amazon Web Service

GW Gateway

IoT Internet of Things

IP Internet Protocol

MN Mobile Node

OMG Object Management Group

RMA Resource Management Architecture

RSSF Rede de Sensores Sem Fio

SDDL Scalable Data Distribution Layer

SGR Sistema de Gerenciamento de Recursos

SLA Service Level Agreement

SoC System On Chip

WSN Wireless Sensor Network

Sumário

1 Introdução

O número de dispositivos de computação usados em tarefas diárias, a cobertura vasta da Internet (Redes Móveis de Próxima Geração) (76) e o modo de tratamento concedido aos dados de rede vêm sendo utilizados cada vez mais com a finalidade de melhorar a vida das pessoas. Essa oferta de serviços de rede e dados de dispositivos da Internet das Coisas (IoT) é a chave para o desenvolvimento de aplicações para os denominados Ambientes Inteligentes (AmI) e Ubíquos (85). Por sua vez, as cidades inteligentes são compostas por ambientes independentes que formam, como resultado, sistemas locais mais simples tais como sensores isolados públicos ou privados espalhados em uma cidade. Esse grande número de componentes abre um problema relevante, relativo a grande quantidade de dados necessárias para essas aplicações, uma vez que as implantações de sensores costumam ter um alto custo. Portanto, o compartilhamento de recursos (dispositivos de IoT) entre os diversos atores deve desempenhar um papel importante neste cenário futuro.

No contexto de sistemas ubíquos (18), as aplicações devem coletar dados de diversos sensores, entretanto, a implantação de uma grande quantidade desses dispositivos pode inviabilizar o projeto. Com o intuito de contornar esta situação, foi desenvolvido um modelo emergente de computação em nuvem chamado sensores como um serviço, do inglês *Sensores as a Service* (*SenaaS*), onde usuários podem publicar os dados de seus próprios sensores que, por sua vez, serão utilizados como dados de consumo para os outros participantes ou consumidores. Desta maneira, os dados se transformam em um serviço da Web consumível, o que incentiva os usuários a compartilharem seus dados como em uma rede de detecção participativa ou, dependendo das circunstâncias, até mesmo a comercialização dos dados como em um mercado de ações. Todavia, além do modelo de negócios, a questão relevante para este trabalho é o modelo de compartilhamento de recursos e sua utilização para o desenvolvimento de sistemas ubíquos.

No entanto, no caso de redes de sensores e sistemas ubíquos, existe uma dificuldade de integração e implantação devido à heterogeneidade de dispositivos e tecnologias de comunicação (103). Neste sentido, é recorrente a utilização de *middleware* como uma

camada que provê abstrações e uma visão homogênea dos recursos e com um foco maior na integração do que na programação (86). No contexto de desenvolvimento de softwares, as melhorias inerentes à utilização de um *middleware* para IoT estão relacionadas à integração de componentes do sistema como hardware, distribuição de dados e tecnologias de comunicação. No geral, o compartilhamento de dados é construído usando um *middleware* executado como uma camada acima dos sistemas operacionais fornecendo, para clientes, uma interface de acesso comum e, para desenvolvedores, um mecanismo que dispensa conhecimento dos acontecimentos nas camadas inferiores. Em relação à heterogeneidade de dispositivos e sistemas, a arquitetura de *software* do *middleware* tem grande contribuição no desenvolvimento de ambientes inteligentes e IoT.

A Internet da Coisas tem o objetivo de ser uma rede global e que, de forma dinâmica, ofereça infraestrutura para que dispositivos físicos ou virtuais (sensores e atuadores) possam se comunicar entre si, mesmo com essa grande diversidade de tecnologias empregadas. Contudo, mesmo os sistemas pervasivos e a IoT criando uma revolução nos novos negócios e na indústria (55), ainda não alcançaram seus devidos potenciais (88). Um dos problemas está relacionado com o processo de desenvolvimento de novos produtos, visto que o paradigma de computação ubíqua muda a forma como os usuários interagem com esses sistemas, alterando o método de desenvolvimento desses *softwares*.

Uma questão notória é o alto custo financeiro de se ter um ambiente pervasivo (5) para a realização de testes, treinamento de novos desenvolvedores e validação de novas propostas. Logo, um modelo bastante utilizado para experimentação na área de redes de computadores e computação de alto desempenho são os *testbed*, laboratórios com os recursos computacionais necessários para determinada aplicação que são compartilhados com a comunidade (25). Os *testbeds* desempenham um papel relevante nos cenários de pesquisa experimental, principalmente por habilitar o compartilhamento de recursos, fornecer um ambiente de experimentação padronizado que ajuda no requisito de reprodutibilidade, pois dispõe dos registros das atividades realizadas e dados a respeito do que se esteja executando. Além disso, os *testbeds* têm crescido por apoiar a experimentação de novas tecnologias, a pesquisa e desenvolvimento de novos serviços e aplicações (91).

1.1 Definição do Problema

No desenvolvimento de sistemas ubíquos exitem muitas questões que o desenvolvedor deve resolver. A primeira delas é como obter os dados. Os sistemas distribuídos tradicionais

diferem de sistemas ubíquos devido ao seu dinamismo. No paradigma ubíquo geralmente se faz uso de contexto, que geralmente é obtido através de sensores e demais dispositivos, geralmente obtidos de uma infraestrutura de IoT.

Desenvolver um sistema ubíquo que suporte um ambiente mais realístico, fazendo uso de RSSF (Redes de Sensores Sem Fio) e demais dispositivos IoT, não é uma tarefa fácil (53). Para atingir os objetivos do projeto o desenvolvedor tem que planejar a implantação e produzir uma arquitetura do sistema e, a partir disso, fazer uso de ferramentas de simulação e experimentos no mundo real que são requeridos para validar seu trabalho (77).

O acesso a sensores e dispositivos IoT pode ser um fator importante na etapa de desenvolvimento. Pesquisadores e desenvolvedores estão sempre precisando de dispositivos IoT para realizar seus testes e implementações, porém construir um ambiente de testes nem sempre é uma opção viável financeiramente para a maioria. E em alguns casos, o dispositivo será utilizado apenas algumas vezes em um curto período de tempo. Uma solução viável poderia ser alugar o dispositivo, ou ainda melhor, usar um dispositivo compartilhado. Por sua vez, esses sensores compartilhados entre os usuários devem ser realizado de maneira automatizada, por exemplo, que o usuário acesse os dados do sensor por um endereço.

A maioria dos laboratórios não tem recursos financeiros para a construção de um ambiente pervasivo ou cômodo inteligente para testes de novos algoritmos. Em contrapartida, existem diversos *testbeds* (44, 2, 25) que disponibilizam uma estrutura padronizada e com diversos sensores, possuindo até mesmo versões em miniaturas dos ambientes em questão tais como maquetes para realização de testes (73, 5). Contudo, a maioria dos *testbeds* apresentam uma estrutura rígida, não sendo possível fazer modificações novos sensores a plataforma. Outra restrição é a de que em *testbeds* maiores e com mais recursos, o acesso tem tempo bastante restrito a usuários externos, muitas vezes com a exigência de processo seletivo para o acesso de pesquisadores externos (91).

Os *tesbeds* de experimentação científica geralmente provêm infraestrutura para implantar, executar e também coletar os dados da experimentação, bem como fornecem *traces* dos dados gerados pelos seus AmI. Todavia caso o desenvolvedor necessite utilizar determinado conjunto de dados de um dos experimentos, o mesmo será responsável por gerar um ambiente próprio para reprodução do experimento, e também por posterior implementação de suas novas ideias e algoritmos.

1.2 Objetivos e Metodologia

Este trabalho tem por objetivo propor de um modelo de arquitetura de serviços que utilize o conceito de *Sensors as a Service* para compartilhamento e gerenciamento de recursos IoT em nuvem. Esses dispositivos sensores e atuadores, deverão estar disponíveis para consumo por aplicações que serão implantadas nessa nuvem. Esse ambiente deverá servir como uma plataforma de experimentação para sistemas ubíquos.

As instâncias dos sistemas de cada usuário deverá possuir um ambiente isolado para realizar seus testes, podendo obter os dados personalizados de dispositivos físicos, virtuais ou até de outros clientes/usuários. Para este fim, o trabalho visa integrar um sistema de gerência de experimentação que permitirá a oferta de serviços de virtualização para tornar os testes mais eficientes e escaláveis, com suporte a reconfiguração, coleção de proveniência e automação de ambientes de testes em computação ubíqua.

Quando considerado o disposto acima listamos alguns objetivos particulares para a arquitetura:

- Desenvolver um modelo que verifique a possibilidade de redução do tempo de implantação das aplicações, com a finalidade de avaliar viabilidade da implantação de aplicações nesse ambiente.
- A arquitetura deve alocar de forma eficiente os containers na nuvem.
- Realizar simulação da arquitetura para avaliar se é factível a implantação.
- Desenvolver um protótipo usando a arquitetura proposta, esse protótipo deve servir para validar como prova de conceito do trabalho.

A partir da definição do problema e dos objetivos apresentados, propomos uma arquitetura para tratar do desenvolvimento de aplicações ubíquas em nuvens, o compartilhamento de dados foi implementado em um modelo de sensores como serviço, foi a forma que se apresentou mais eficiente para fazer a integração e compartilhamento de dados de dispositivos dentro do ambiente.

Para validar a arquitetura, foi elaborado uma simulação dividida em duas etapas, modelagem com objetivo de fornecer um ambiente que consiga fornecer uma utilização eficiente de recursos e implantação de forma mais eficiente. Primeira etapa de formulação matemática acerca alocação de recursos e a redução do tempo de implantação das

aplicações, e uma segunda etapa a simulação desses modelos em um simulador de nuvem para verificar o desempenho. Seja em uma nuvem pública ou privada, essas informações são relevantes tanto da questão do operação quanto de custo financeiro. Uma prova de conceito foi implementada para a corroborar com o que foi desenvolvido, um sistema ubíquo rodando em uma nuvem local que controla um protótipo de casa inteligente com diversos dispositivos, tudo implementado seguindo os princípios das abstrações da nossa arquitetura.

1.3 Estrutura da Tese

A tese está estruturada da seguinte maneira. Capítulo 2 é apresentado alguns temas, tecnologias, ferramentas e definições relevantes para melhor entendimento do trabalho. O Capítulo 3 apresentamos a revisão da literatura, considerando os sistemas que são importantes para o arranjo da arquitetura, como os *middlewares*, utilização de virtualização em experimentos, *testbeds* e a disponibilização/compartilhamento de dados dos experimentos. Nós apresentamos a proposta de uma arquitetura no Capítulo 4, seus componentes e questões de projeto de implementação. Já o Capítulo 5 trata da avaliação da arquitetura por meio de uma modelagem, e com essa base foram realizadas simulações da proposta e ao fim uma prova de conceito é apresentada. E finalmente no Capítulo 6 serão expostas as considerações finais, bem como as limitações do trabalho e futuras direções.

2 Referencial Teórico

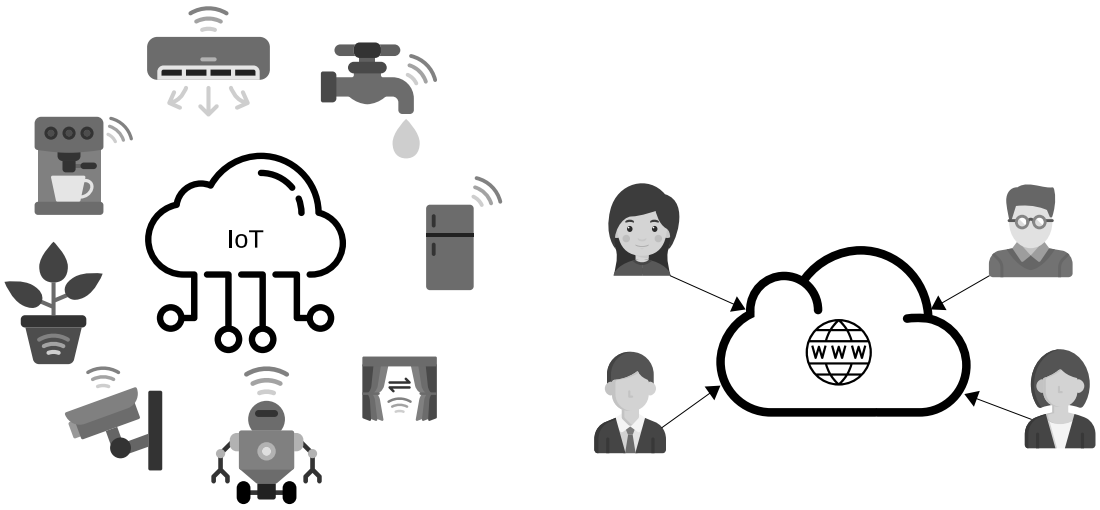
Neste capítulo vamos apresentar os conceitos básicos para os principais temas pertinentes a este trabalho. Na Seção 2.1 é introduzido o tema de Internet das Coisas e sua fundamentação. Na Seção 2.2 é discutido a função do *middleware* e suas características. A Seção 2.3 trata do paradigma de *Sensors as a Service* e o funcionamento de suas camadas. Por fim, a virtualização e utilização de contêineres é comentada na Seção 2.5.

2.1 Internet das Coisas

Quando do projeto das primeiras redes de computadores, os protocolos foram imaginados para atender os parques computacionais da época, que se baseavam nos, até então, escassos computadores dos centros de pesquisas. Essas redes eram estacionárias, e respeitavam a premissa de que todos os nós deveriam ter acesso entre si. Porém com a popularização do computador pessoal e o crescimento da Internet, ficou claro que o protocolo IP (*Internet Protocol*) se mostrou despreparado para acompanhar os avanços tecnológicos uma vez que não foi projetado para operar com uma quantidade tão grande de dispositivos. De acordo com Gardner (61), em um futuro próximo iremos ter cerca 20 bilhões de "coisas"conectadas na Internet e em 2030 a projeção é que sejam 100 bilhões.

O termo *Internet of Things* surgiu em 1999 quando Kevin Ashton (8) descreveu o sistema de identificação por rádio frequência (RFID) para os executivos da P&G (Procter & Gamble). Segundo a definição de Asthon, a Internet, como a maioria das pessoas conhece, funciona como uma rede de sensores, onde esses sensores são as pessoas, pois elas é que alimentam a rede com os dados. Já essa nova Internet é formada por diversos sensores, ou unidades computacionais, que são os objetos do nosso dia-a-dia, só que com capacidade de processamento e comunicação (janelas, torneiras, geladeiras entre outros). Esses dispositivos, agora interligados em rede fazem uma nova internet, a Internet das coisas, a Figura 1 denota essa comparação.

Mark Weiser em seu trabalho seminal (104), definiu a computação ubíqua como um



(a) *Internet of Things* (b) *Internet of People*

Figura 1: Visão de Kevin Ashton para a Internet das Coisas

método para melhorar o uso dos computadores, os tornando disponíveis no ambiente físico, mas de forma invisível para o usuário. De forma adicional, Weiser afirmou que os computadores iriam desaparecer uma vez que seriam incorporados aos objetos do dia a dia. Neste contexto, os assim chamados sistemas ubíquos serão alimentados por dados de uma grande quantidade de objetos. Os usuários cada vez mais terão dispositivos acoplados no seu corpos, a chamada computação vestível (*wearable computing*), presente em vestidos e roupas, ou em relógios e óculos. Esses dispositivos devem funcionar sem a interferência do usuário, em funções especializadas, como por exemplo o crachá de localização desenvolvido por Want (102) que emite um sinal periodicamente para informar a localização do usuário. Os sistemas ubíquos devem modificar o seu comportamento baseado em dados coletados do ambiente, por exemplo, o crachá de localização pode informar aos dispositivos próximos a presença de uma pessoa específica, e a partir dessa informação modificar a temperatura do ambiente, destravar uma porta ou até modificar a iluminação baseada do gosto do usuário. Essas características fazem parte de uma subárea importante que é a computação sensível ao contexto (87).

2.2 Middleware

Middlewares são aplicações ou uma camada de *software* que visa facilitar para os desenvolvedores a implementação da troca de mensagens. Ademais, também é uma tecnologia que permite a integração entre diferentes *softwares* que rodam em diferentes sistemas e dispositivos (6). Com a expectativa de crescimento do número de dispositivos, os *middlewares*

terão um papel fundamental na questão de integração e comunicação entre sistemas. Por este ângulo, eles devem suportar a escala de crescimento e a grande massa de dados proveniente dessas redes IoT.

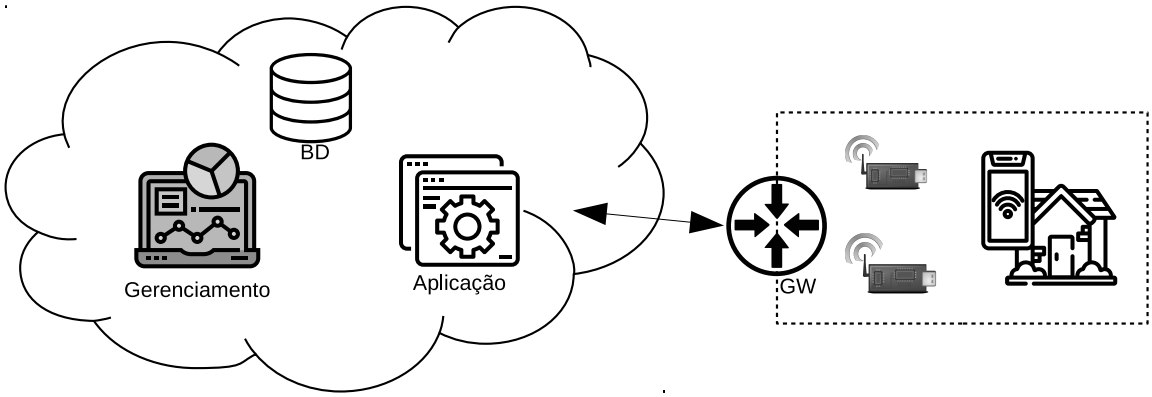


Figura 2: Infraestrutura IoT Baseada em Nuvem

Como visto na Figura 2, hoje os *frameworks* de IoT se baseiam em um conjunto de sensores e atuadores ligados a uma nuvem por um *gateway* IoT. Por sua vez, nessa nuvem é que a aplicação vai trabalhar com o processamento dos dados desses sensores, ou seja, o *middleware* deve abstrair a complexidade dos diferentes tipos de dispositivos e fornecer uma interface que permita os desenvolvedores tenham a autonomia para focar na aplicação e interação com os usuários. No mais, o *middleware* tem ainda que atender aos seguinte requisitos:

Gerenciamento de Dispositivos: Cada dispositivo que entra na rede deve ser identificado e inicializado. Dado que os dispositivos são voláteis, o sistema tem que tratar das falhas físicas que os dispositivos apresentem, como o mal funcionamento por problemas de comunicação como desconexão e variação na taxa de transmissão.

Escalabilidade: Todos os sistemas de Internet das Coisas devem possuir a capacidade de atender o crescimento da quantidade de dispositivos que é capaz de gerenciar e a quantidade de dados que consegue manipular nos processos de comunicação, armazenamento e processamento.

Privacidade e Segurança: Os usuários de qualquer sistema precisam ser protegidos, quanto ao seus dados, visto que os ambientes ubíquos podem conter muitas informações sensíveis. Neste sentido o acesso não autorizado a um sistema que controla diversos atuadores pode causar sérios problemas.

Mobilidade: A mobilidade é um caso especial nos ambientes inteligentes, visto que o usuário pode se mover entre diferentes espaços físicos, bem como os dispositivos

também pode se mover, havendo a necessidade de tratar a questão de desconexão, troca de rede e também a troca de contexto por mobilidade entre espaços.

Eficiência Energética: Quando tratamos de dispositivos físicos, temos que ter em mente as limitações que seus *hardwares* fornecem. Em alguns casos, esses elementos computacionais vem com baterias acopladas e seu uso deve ser feito respeitando a economia desse recurso. Outro ponto refere-se ao consumo de energia devido ao processamento de uma grande quantidade de dados; essa limitação também deve ser atendida por *middleware* de IoT.

2.2.1 ContexNet

O *middleware* ContextNet é um projeto do LAC (*Laboratory for Advanced Collaboration*) da PUC-Rio que tem o objetivo de fornecer serviços de contexto para aplicações pervasivas. O *middleware* conta com comunicação de dados em tempo real, possui a capacidade de inferir eventos baseado no processamento de *streams* de dados, para isso utiliza *Complex Event Processing* (CEP).

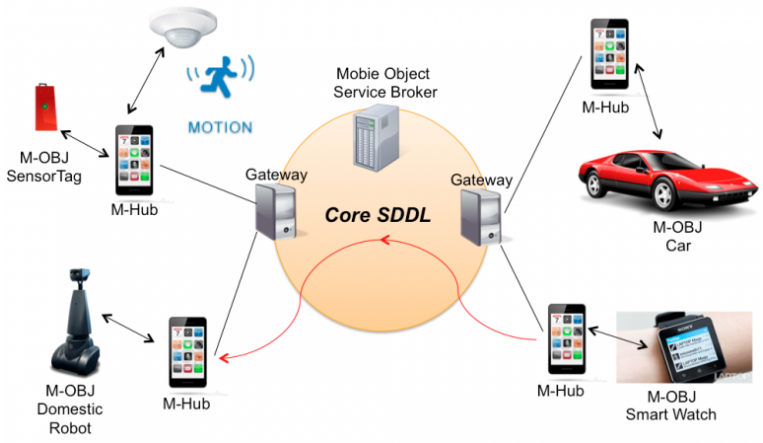


Figura 3: ContextNet e SDDL [LAC-PUC] (28)

O ContextNet, como podemos observar na Figura 3, tem uma camada de comunicação e distribuição de dados chamada SDDL (Scalable Data Distribution Layer) (97) que é utilizada para ligar o núcleo da rede, bem como seus nós móveis (*MN - Mobile node*). Em geral, usa dois protocolos para realizar a troca de mensagens, o RTPS (Real-Time Publish-Subscribe) que como o próprio nome diz uma o modelo de *publish/subscribe* e é usado nas comunicações com fio. Por sua vez, as comunicações entre o núcleo da rede e os MN é realizada utilizando um protocolo próprio chamado MR-UDP. Este protocolo é responsável pela abstração na comunicação, tratando, desta maneira, questões de desco-

nexão, reconexão e troca de rede. Outros serviços e funções podem ser construídas em cima desta camada.

Outro recurso disponibilizado pelo ContextNet é o Mobile Hub, que é um *middleware* que possibilita o uso de dispositivos Android como infraestrutura de rede para permitir acesso aos Mobile Objects (M-OBJs sensores ou atuadores) que não tenham interfaces IP. Esses dispositivos, apresentam interfaces WPAN de curto de alcance (exemplo: RFID, NFC, Bluetooth e ZigBee). Quando um dispositivo Mobile Hub se aproxima, ele faz a leitura dos dispositivos ao alcance e encaminha para o núcleo da rede.

2.2.2 FIWIRE

A crescente aceleração do desenvolvimento tecnológico em todas as áreas da arquitetura de sistemas de informação está se espalhando rapidamente para a Internet das Coisas. Muitas plataformas de IoT para fins gerais e especiais já estão disponíveis. O FIWARE (27) é uma plataforma que está na fase de transição de pesquisa acadêmica para um modelo de exploração comercial. A ideia do FIWARE baseia-se em eventos reais, no mercado global de IoT, Smart Cities e Veículos Autônomos e o FIWARE atua para prover serviço de PaaS para essas finalidades. Esse modelo de plataforma como serviço foi construído com a premissa de integrar diversos cenários de soluções em cidades inteligentes, como gerenciamento de águas, mercado IoT, Carros autônomos, operação de tráfego entre outros.

O FIWIRE tem um foco em operação de cidades inteligentes, e de uma solução aplicada em determinada cidade possa ser aproveitada por outras, trata da organização e integração de sistemas legados de IoT aplicados em Smart Cities, traduzindo e estruturando dados desses sistemas em um sistema de federação. Como o sistema atua em solução de cidades inteligentes não é considerado para suporte para aplicações ubíquas, mesmo com o mapeamento de gerenciamento de dispositivos e contexto.

2.3 Paradigma Sensor as a Service

Sensor as a Service (*SenaaS*) não é uma ideia nova (50) (107), visto que entra no modelo de *Everything as a Service* (38, 9) que popularizou-se junto com as tecnologias de nuvem. Já existem muitos trabalhos que implementam a distribuição dos dados de redes de sensores usando tecnologias *Web*. Porém, mais recentemente, o emprego desse modelo vem sendo difundido junto a um novo paradigma, que descreve o consumo de dados de ambi-

entes ubíquos em um modelo que estimula o compartilhamento dos dados e, em alguns casos, a venda dos mesmos como em um mercado de ações (80).

Em um ambiente cada vez mais pervasivo, as pessoas serão estimuladas a compartilhar os seus dados. Por exemplo, em uma casa com dados gerados pela geladeira, máquina de café, fogão e outros, existe um grande interesse comercial nessas informações. Deste modo, uma rede de supermercados poderia utilizar essas informações e melhorar a oferta de produtos, baseado no consumo observado nos bairros próximos. Baseado na Figura 4, o modelo de *SenaaS* pode ser dividido em três categorias, Sensores e Donos de Dados, Provedores de Dados e Consumidores de dados. A seguir definimos cada uma delas:

Sensores e Donos de Dados: Podem ser pessoas, empresas públicas ou privadas. São os detentores dos sensores, decidem se querem enviar suas informações pra nuvem ou não. Para disponibilizar seus dados precisam de um provedor de dados, que será informado sobre quais os dados que podem ser disponibilizados e sobre a possível cobrança de taxas. Fica a cargo do dono a decisão sobre se a disponibilização ser gratuita ou não. Também por critério de privacidade o dono tem o poder de decidir se um dado pode ser publicado ou não, nenhum provedor pode publicar um dado o qual dono não tenha autorizado.

Provedores de Dados: Os provedores de dados fazem um contrato com os donos de dados. Nesse contrato as partes acordam sobre quais dados serão disponibilizados, assim como o acordo de privacidade, segurança e valores financeiros das transações. Os provedores após obterem a permissão dos donos, são os responsáveis de ler os sensores e publicar seus dados na nuvem. Após essa etapa o provedor espera até que alguém consulte por esse dado.

Consumidores de Dados: Os consumidores são os interessados pelos dados e se registram nos provedores para obter o acesso. Fazem parte dessa classe os pesquisadores acadêmicos, empresas, indústrias e também o governo. Os consumidores não obtêm acesso ao sensor de forma direta, eles sempre fazem o acesso através do Provedor de Dados, o único que possui acesso direto aos sensores.

Esse paradigma pode ser muito útil em cenários onde exista o interesse em compartilhamento de recursos IoT, principalmente quando se trata de ambientes de testes ou laboratórios de pesquisa. Sem a necessidade de cobrança de taxas, o modelo *SenaaS* pode ser a base para um modelo de colaboração de sensores.

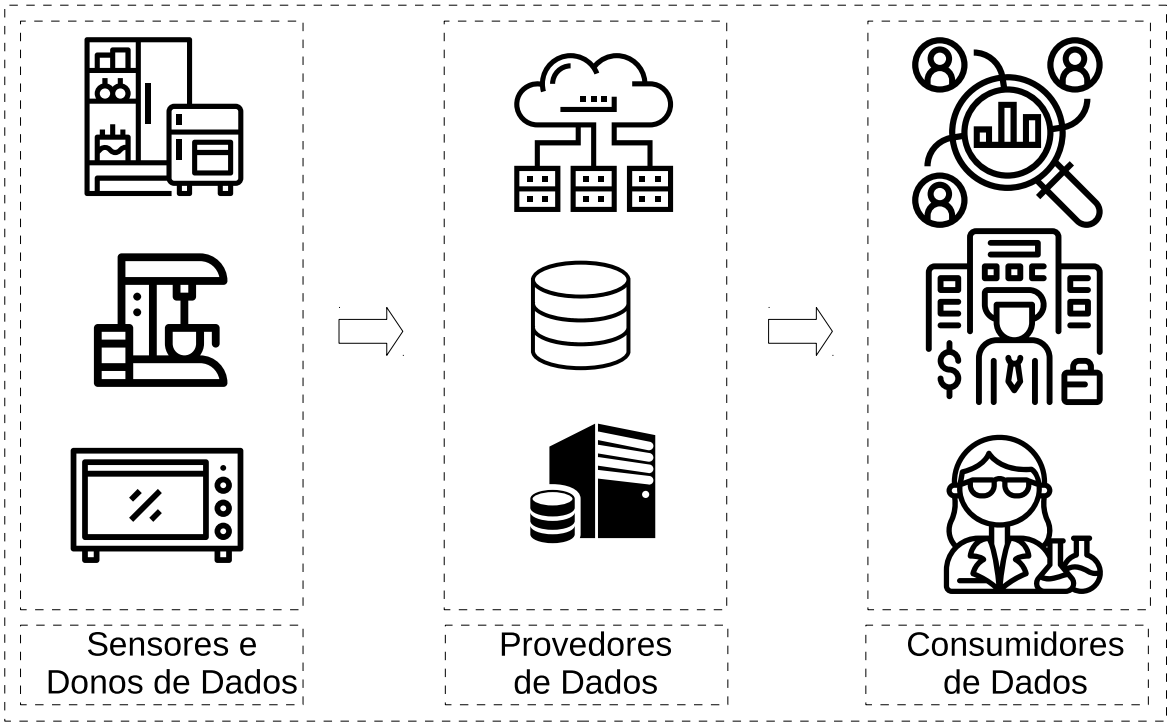


Figura 4: Modelo do *Sensor as a Service*

2.4 Modelos de Comunicação

Os tipos mais comuns de sistema de mensagens são do tipo ponto a ponto. Nesse sistema as mensagens persistem na fila, e um ou mais clientes podem consumir mensagens paralelamente. Por outro lado, uma mensagem especifica só pode ser lida uma vez e por apenas um consumidor. A Figura 5 representa esse mecânismo. Já no sistema *pub/sub* (*Publish/Subscribe*), as mensagens são persistidas em tópicos. Os consumidores pode se inscrever em mais de um tópico e consumir todas as mensagens relativas a eles. Os produtores das mensagens são chamados de publishers (publicadores) e os consumidores de *subscribers* (assinantes) (42).

Muitos sistemas de monitoramento tem a necessidade de entender o que se passa no ambiente em uma taxa frequente e periódica. Para os sistemas ubíquos, as leituras de dados periódicas são usualmente executadas durante todo o tempo de execução do sistema, por exemplo, monitoramento da saúde de um idoso. Assim a maneira que é mais utilizada para esses tráfego de dados é *data streams*. Os primeiros modelos de arquitetura que se valiam de *data stream* utilizavam uma abordagem que não se aplica a maioria dos sistemas atuais, onde o dado inserido dentro do sistema tem valor apenas em um curto espaço de tempo e alimenta seus consumidores em tempo real e depois é descartado. Nas

abordagens modernas esses dados são armazenados em banco de dados para utilização em consultas de mensagens já passadas.

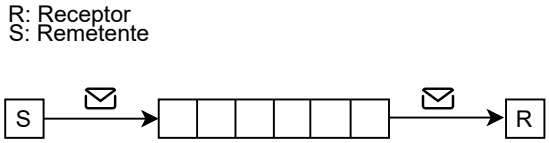


Figura 5: Mensagem ponto a ponto.

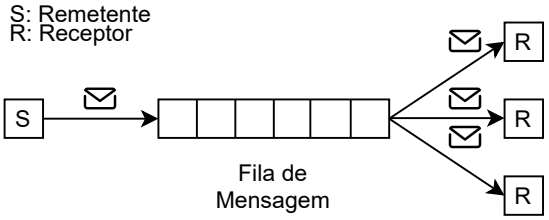


Figura 6: Mensagem no modelo Publish-Subscribe.

2.5 Virtualização e Contêineres

As grandes empresas de internet como Google, Amazon e Microsoft usam sistemas de nuvem computacional, construídas geralmente em cima de ferramentas de virtualização. Algumas destas empresas inclusive oferecem a virtualização como serviço, são os chamados *Container as a Service* (52), que funcionam em conjunto com o *Platform as a Service* (*PaaS*) (81), do qual é responsável por gerenciar as dependências das aplicações, e o *Infrastructure as a Service*, que controla os recursos de rede, recursos físicos entre outros (16).

Existem diversas ferramentas para a criação e gerenciamento de contêineres, com as mais diversas funções. Contêineres são uma forma de isolamento e gerenciamento de ambientes Unix. Vale ressaltar que sistemas operacionais com esta arquitetura fornecem alguma forma de conteinerização e, por este motivo, disponibilizam uma maneira eficiente de isolar algum processo do resto do sistema. Esses esquemas fornecem o mesmo grau de separação e isolamento que os modelos de virtualização completa.

A Figura 7, apresenta o comparativo entre os modelos de virtualização completo ou tradicional e a virtualização por contêiner. No modelo tradicional temos o sistema operacional com um hipervisor responsável por rodar instalações completas de máquinas virtuais (VM) de um sistema operacional qualquer. O modelo de contêiner carrega apenas as bibliotecas e dependências necessárias a sua aplicação, compartilhando o *kernel*

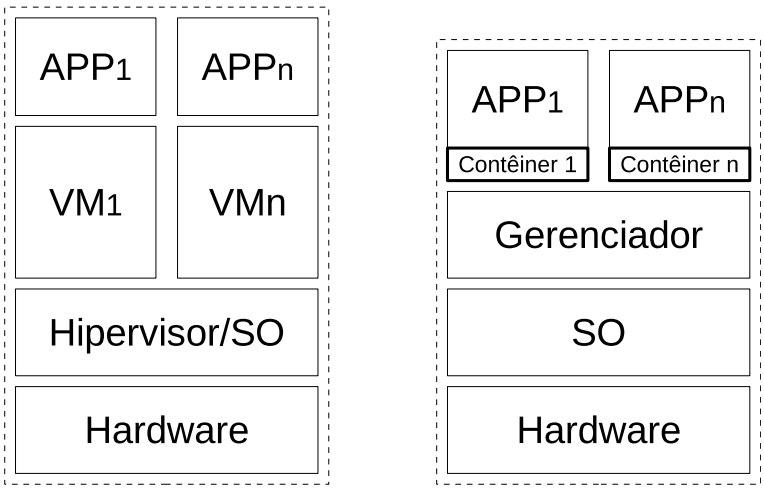


Figura 7: Virtualização completa X Virtualização com Contêineres

e demais estruturas pertencentes ao sistema operacional (37). Devido a tal característica, os contêineres apresentam um tamanho e consumo de recursos reduzido (*lightweight virtualization*) (98).

Os principais sistemas de conteinerização utilizados em trabalhos científicos ou na industria são o Docker (36), o Linux Container LXC (84) e o Kubernetes (48) que é oferecido pelo Google. A seguir apresentamos uma breve descrição dos mesmos.

2.5.1 LXC

Linux Container LXC é uma ferramenta poderosa de virtualização, pois apresenta uma grande portabilidade, dado que não necessita de emulação de *hardware* e pode rodar uma instância (*lightweight*) do sistema operacional sem a sobrecarga de uma instalação completa sobre um Hipervisor. O LXC usa o `cgroups` (*Control groups*) como interface para o *kernel*, controlando, deste modo, os recursos de isolamento de processos e rede. Esse tipo de virtualização promete um desempenho similar ao de uma aplicação nativa (15).

2.5.2 Docker

O Docker (36) é uma ferramente de código aberto que oferece uma maneira de automatizar de forma fácil a implantação de serviços de aplicações Linux, utilizando contêiner para isso. Ele é baseado no LXC (17), roda os processos em isolamento tanto de memória e UCP quanto rede e interfaces de entrada e saída. Seu funcionamento baseia-se na criação de imagens que incluem seu próprio *namespace*, hierarquia de processos e sistema de arquivo. Essas características permitem ao Docker a sua adoção em reprodutibilidade

nas pesquisas, pois a maneira sofisticada com que gerencia seus pacotes e configuração de ambientes podem fornecer uma padronização na montagem de experimentos, bem como o compartilhamento de imagens ou migração de experimentos entre diferentes sites (17).

2.5.3 Kubernetes

O Kubernetes (48) foi desenvolvido pelo Google é um orquestrador de implantação de aplicações conteinerizadas. Para alguns usuários, a utilização de apenas um contêiner pode ser o suficiente. Porém para a utilização em larga escala que o Kubernetes tem seu maior impacto, uma vez que consegue controlar um ambiente de contêineres espalhados por várias máquinas, fornecer um monitoramento do funcionamento, capacidade de replicação além de facilitar a comunicação entre seus ambientes virtuais. Outra característica interessante é que ele pode agrupar vários contêineres em grupos chamados *pod*, que possuem um IP único que pode ser acessado por qualquer usuário dentro da rede mesmo não estando na mesma máquina (14).

2.6 Discussão

Neste capítulo foi apresentado o referencial teórico para aplicação na tese, visto que os sistema ubíquos e de ambientes inteligentes geralmente são baseados em dispositivos IoT, a comunicação destes equipamentos se dá usualmente por WiFi, e o modelo de comunicação é gerenciado através de *middlewares* que usam sistemas de *publish/subscribe* e *stream* de dados.

Praticamente toda plataforma de computação em nuvem atualmente emprega as tecnologias de virtualização a nível de sistemas operacionais, fazendo o uso de contêineres, basicamente as aplicações são isoladas em nesses ambientes e rodam de maneira a fornecer alta resiliência, pois podem ser executados e realocados de maneira rápidas com controle de falhas. Isso pode ser controlado através de orquestradores, que são os responsáveis por organizar a execução de vários contêineres bem como gerir sua execução. Por esses e outros motivos, esse trabalho faz uso de contêineres e pode empregar modelos de diversos distribuidores.

Uma arquitetura de *software*, pode ser definida como uma abstração de todos os elementos em execução de um sistema computacional, ela é definida como uma forma de esconder determinadas particularidades (89) com intuito de melhor dividir suas propriedades e funções. A divisão da arquitetura em níveis ou camadas que se comunicam por

interfaces que interagem entre si em determinado nível (12). Este trabalho vai fornecer uma arquitetura de software que irá combinar as tecnologias apresentadas nesse capítulo de maneira a atender as necessidade da proposta.

3 Trabalhos Relacionados

As pesquisas em Ambientes Inteligentes (AmI) e computação ubíqua têm buscado estratégias tecnológicas para melhorar a qualidade de vida das pessoas e resolver problemas causados pelo crescimento populacional em áreas urbanas. Muitos artigos falam sobre as tecnologias para melhorar os serviços para as pessoas e também agregar valor à serviços de empresas públicas ou privadas (69). Nestes AmI, teremos dados provenientes de pessoas, redes sociais, lares e organizações (públicas e privadas), diferentes obras abordam muitos tipos de problemas relacionados a estes sistemas pervasivos (31) e que tem semelhanças com o foco do nosso trabalho. Outros trabalhos tratam complexidade de distribuição de dados nesses AmI além da complexidade de implementação dos nós IoT em diferentes plataformas (1), o que leva a adoção de ferramentas de *software* como *middlewares* para abstração dessas complexidades, abordagens mais tradicionais de acesso as informações como *web services* também são bastante utilizadas.

Em trabalhos mais recentes a preocupação com a questão de reprodutibilidade de experimentos e proveniência dos dados de testes científicos, acabaram por incluir os requisitos de isolamento de recursos, tráfego de dados entre outras exigências, esses requerimentos que são discutidos em trabalhos de tecnologias virtualização para aplicações em nuvem. Muitos projetos de pesquisa exploram a utilização de *testbeds* de ambientes reais para aplicar seus trabalhos, outros utilizam técnicas de simulação ou construção desses locais em escala reduzida, neste capítulo vamos apresentar alguns desses projetos.

3.1 Middlewares de Distribuição de Dados

O middleware GSN (1) (Global Sensor Network) é projetado para criar uma maneira fácil de programar redes de sensores e seguir com base nessas características: simplicidade, adaptabilidade, reconfiguração e escalabilidade, além do baixo uso de recursos. Implantação em uma plataforma única para integração de múltiplas redes de sensores, podendo ser utilizada para grande número de sensores heterogêneos. A principal característica do

GSN é a abstração do sensor usando descritores XML e consultas SQL para recuperar informações. O GSN também conta com uma forma de publicação de seus dados utilizando *Web Services* para representar os dados dos seus sensores. No *middleware* cada sensor é representado usando um *wrapper* que especifica quais protocolos e configurações de rede, qual o procedimento para leitura de dados do sensor e o que fazer com os dados lidos do sensor. Processamento de *Data Stream* tem recebido grande atenção recentemente, pois fornecem a detecção de eventos complexos que podem ser muito úteis na geração de contexto, o GSN faz a manipulação dessas *streams* utilizando consultas SQL.

O ContextNet (40) é um serviço para fornecer dados de contexto em redes estacionárias e móveis. Conta com uma camada de distribuição de dados escalável (*Scalable Data Distribution Layer* - SDDL) (97), pode ser utilizado para aplicações que necessitem de mobilidade, e comunicação entre grupos (34). Esse serviço trata de questões importantes no que diz respeito a comunicação de dados, como tolerância a falhas, balanceamento de carga de rede, suporte a desconexão de nós e segurança. Ele também fornece recursos de criação e gerenciamento dinâmico para grupos. O middleware ContextNet funciona em um modelo de publish/subscribe. A transferência de dados ocorre usando dois protocolos: o MR-UDP (92) e o OMG DDS (78). O MR-UDP trata mensagens entre um cliente e um *gateway*; o DDS é responsável pela distribuição de dados no núcleo da rede. Usando o ContextNet, é possível alcançar o crescimento de uma rede, garantindo a escalabilidade da distribuição de conteúdo entre uma grande quantidade de dispositivos.

Outro trabalho que usa a abordagem de *Web Service* é o Tiny Web Services, eles apresentam um sistema interoperável que oferece uma interface de linguagem de descrição de serviços da Web e é projetado para os usuários acessarem os sensores. A principal característica deste método é o foco em nós sensores sem fio de baixa potência e perfil de energia desta implementação.

O middleware SOCRADES (35), é uma solução para integração de negócios que pode ajudar nos processos de produção em fabricas, oferecendo uma arquitetura orientada a serviços da web que integra diferentes tipos de objetos e máquinas inteligentes. Este trabalho se concentra em dispor esses serviços da Web de forma acoplada a uma solução de negócios corporativos como um ERP (*Enterprise Resource Planning*).

Uma interface para fornecer aos usuários a capacidade de construir serviços baseados em objetos físicos, sejam eles sensores ou atuadores, usando o conceito *Web of Things* (46), no qual os objetos do mundo real são integrados como aplicativo *REST*. O Webplug (72) descreve uma estrutura que reconstrói essas ideias usando uma abordagem baseada em

URL e pode ser implantada na nuvem porque depende do mecanismo HTTP.

Trabalho	Escalável	Orientado a Serviço	API	Comunicação de Eventos
AmbieAgents (57)	Sim	Sim	Não	Não
CAMPS (82)	Sim	Sim	Sim	Query/Subscribe
CAMUS (70)	Não	Sim	Não	Não
Cobra (24)	Não	Sim	Não	Não
CoCaMAAL (43)	Sim	Sim	Não	Data Stream
CONASYS (62)	Não	Sim	Sim	Não
ContextNet (41)	Sim	Sim	Sim	Publish/Subscribe
Gaia (83)	Não	Sim	Sim	-
GSN (1)	Sim	Sim	Sim	Data Stream
SOCAM (45)	Não	Sim	Sim	Não
SOCRADES (35)	Sim	Sim	Sim	Publish/Subscribe
UbiRoad (95)	Sim	Sim	Não	Não
Webplug (72)	Não	Sim	Sim/REST	Não

Tabela 1: Middlewares aplicados a sistemas sensíveis ao contexto.

O AmbieAgents (57) é um sistema de infraestrutura escalável para ambientes de usuários móveis, que entrega informações de contexto. Foi implementado em um ambiente real, no aeroporto de Oslo. Fornece contexto como um serviço, porém não tem um modelo de comunicação que compartilhe dados ou uma API. Já CAMPS (82) é um *middleware* para espaços inteligentes, baseado em sistemas de multi-agentes. É focado em fornecer contexto através de ontologias e tem uma implementação de protótipo de uma sala de aula inteligente.

No trabalho de Hung (70) (CAMUS), é um middleware que foi aplicado no desenvolvimento de um *testbed* de uma casa inteligente. CAMUS fornece interfaces para comunicação com soluções de outros *middlewares*. Esta comunicação entre plataformas permite o compartilhamento de dados e adiciona uma maior quantidade de informações de contexto disponíveis para os usuários do CAMUS.

Cobra (*Context Broker Architecture*) é um arquitetura que foi desenvolvida como parte da tese Chen (24), ela é baseada em agentes que oferecem suporte a sistemas sensíveis ao contexto em ambientes inteligentes. Parte integrante dessa arquitetura é a verificação ativa do contexto, que mantém um modelo compartilhado do contexto para uma comunidade de agentes, um protótipo de demonstração da arquitetura foi desenvolvido e aplicado a um sistema de salas de conferências inteligentes chamado *EasyMeeting* (23).

O CoCaMAAL (43) (*Cloud-oriented Context-aware Middleware in Ambient Assisted Living*) se propôs abordar questões como a complexidade de gerenciamento de dados de

sensores e informações de contexto, monitoramento de atividades humanas e na descoberta de serviços. Estes são problemas clássicos no que diz respeito a ambientes inteligentes e ambiente de vida assistida (AAL - *Ambient Assisted Living*). O CoCaMAAL implementa uma arquitetura orientada a serviço baseado em nuvem, onde também fornece uma API para raciocínio de contexto.

CONASYS (*Context Aware System*) é um sistema com a capacidade prover informações sensíveis ao contexto a entidades de ambientes de IoT (62). Fornece uma API para os usuários e juntamente com o middleware COMPaaS (4) (*Cooperative Middleware Platform as a Service*), que facilita a criação de aplicações IoT por fornecer uma infraestrutura para os dispositivos.

Gaia é uma infraestrutura de middleware experimental que foi utilizada para prototipar o gerenciamento de recursos e fornecer as interfaces orientadas ao usuário para espaços físicos que possuem recursos de computação, especificamente uma sala de reuniões (83). O middleware tem recursos para consultar e utilizar recursos computacionais existentes, com ele é possível acessar os contextos do ambiente.

O SOCAM (*Service-Oriented Context-Aware Middleware*) foi desenvolvido para a construção e prototipagem rápida de serviços sensíveis ao contexto (45). Ele fornece suporte para adquisição, descoberta, além de permitir o acesso a vários contextos. O sistema descreve um modelo de contexto e uma arquitetura de middleware. Também é apresentado um estudo de desempenho para um protótipo em uma casa inteligente.

Na Tabela 1 é feita uma sumarização de alguns trabalhos relevantes para a construção de sistemas ubíquos. Nela anotamos algumas características que julgamos importantes para a construção desses sistemas. A escalabilidade é sempre uma característica importante para *middlewares*, pois ela define a capacidade de inclusão de novos dispositivos mantendo o desempenho. O *middleware* deve permitir o fornecimento de interface de programação de aplicativos (API) para abstrair as complexidades do sistema. *Middleware* orientado a serviço deve fornecer serviços relevantes de acordo com os requisitos do usuário.

3.2 Experimentação com Ambientes Virtualizados

Nesta seção, nós apresentamos algumas propriedades que devem estar disponíveis em ferramentas de gerenciamento de experimentos em sistemas distribuídos baseado em pesquisa na literatura. Virtualização a partir de contêineres ao contrário de tecnologias de hiper-

visor, ou monitor de máquina virtual, não rodam em um sistema operacional completo, mas sim de uma parte autocontida do sistema e com acesso ao *kernel* do anfitrião.

As tecnologias modernas fornecem uma forma de aproveitar os recursos entre diferentes máquinas virtuais, e vem se consolidando como tecnologia base para a implementação de *testebeds*. No trabalho de Huang (49) são descritos alguns requisitos e características para a construção de *testbeds* usando virtualização, são elas, isolamento, escalabilidade, contenção e extensibilidade. A questão de isolamento diz que os sistemas de *testbed* devem em sua construção fornecer a característica de que experimentos em paralelo não interfiram entre si, bem como não haja a degradação devido a outras execuções simultâneas, isto pode ser algo crítico a respeito desses sistemas. A escalabilidade representa a capacidade do *testbed* de aumentar a quantidade de carga e seu crescimento de trabalho. Contenção é o recurso que ambientes virtualizados devem apresentar para que interferência de sistemas externos não comprometam ou alterem o resultado de algum experimento. Já reprodutibilidade deve fazer parte de qualquer experiência científica e não deve ficar de fora de um *testbed* cientifico.

Proposto por Asaeda (7), o CUTEi (*Container-Based Unified Testbed*) é um *testebed* baseado em contêiner que foi projetado para construir os nós usando virtualização. Ele possibilita os usuários executarem experimentos na área de redes centradas em conteúdo, o sistema foi implantado em nove laboratórios ao redor do mundo, foram realizados testes de performance que quando comparados ao PlanetLab(26) mostraram que o CUTEi apresentou melhor estabilidade. A arquitetura do sistema conta com tecnologia de virtualização Linux Container LXC (14), os autores descrevem que os usuários podem usar as versões de contêineres pré-instalados e configurados, bem como fazer a utilização de suas próprias instalações com seus próprios protocolos, desta forma personalizando ainda mais seus experimento. O *testebed* foi construído em cima da implementação do protocolo CCNx (51), uma de suas principais contribuições para foi a robustez e facilidade de utilização quando comparado com soluções pré-existentes, demonstrando que a facilidade de utilização pode ser uma motivação para a criação de novos *testbeds*.

Vlögler apresenta LEONORE (99), que é uma arquitetura que traz três componentes principais que são os Pacotes de Aplicações, *Gateways* IoT e o *framework* propriamente dito. Abordagens mais tradicionais se baseiam em arquiteturas de camadas, onde a camada mais baixa é responsável pela comunicação com os dispositivos IoT na borda (3), geralmente usando um *middleware* para unificar a maneira de acesso das camadas mais superiores, a lógica e inteligência das transações ficam assim restritas as camadas supe-

riores e os nós da borda devem funcionar de uma maneira mais reativa supondo que já exista um *firmware* que permita esse modelo (59, 32). O que geralmente acontece é que os dispositivos nem sempre são pré-configurados, necessitam de ajustes manuais, ou até mesmo implementações especificas para cumprir com seus propósito. Com a possibilidade de utilização de dispositivos cada vez mais potentes computacionalmente (Intel Galileo e Raspberry Pi) nesses ambientes, eles tem capacidades além de apenas sensoreamento, esses recursos não utilizados podem ser aproveitados ao sistema IoT para dividir a carga de trabalho ou até mesmo resolver parte do processamento na borda (IoT *Gateway*).

Outra questão é o componente Pacote de Aplicações, o LEONORE armazena pacotes instaláveis de *software* no servidor e o *framework* realiza o orquestração do provisionamento dos pacotes. Os experimentos do trabalho foram realizados em um ambiente de nuvem, onde os *gateways* (dispositivos IoT) usavam imagens (*snapshot*) do Docker que são utilizadas em dispositivos físicos. Para validar o projeto eles precisavam de vários milhares de nó *Gateways* e a utilização de contêineres possibilitou esse objetivo por serem mais leves que as máquinas virtuais tradicionais e com uma sobrecarga muito baixa.

No trabalho de Wang, foi proposto uma arquitetura de processamento intra-rede IoT para gerenciamento de serviços (101). O trabalho se concentra na apresentação CS-Man que é um serviço que usa os conceitos de NFN (*Named Function Networking*) (90) para realizar escalonamento das tarefas de redes IoT, desta forma a rede é capaz de calcular tarefas específicas para determinado nó da rede realizar, bem como a partes de informações que serão necessárias. O protocolo NDN (*Named Data Networking*) (108), utilizado em redes orientadas a conteúdos, também é aplicado no CS-Man, com isso permite o suporte de *cache* através das rotas entre os nós da rede, melhorando o tempo de entrega de dados repetidos na rede. Para a realização de experimentação da proposta os autores utilizaram uma simulação, utilizando contêineres no Docker para funcionarem como os nós da rede, foram usados cinco cenários de testes que mostraram a viabilidade da abordagem.

A utilização de *Cloud Computing* em conjunto com IoT, geralmente trata-se de um sistema na nuvem que serve de plataforma para conectar objetos inteligentes. Celesti (21) explora o uso de nuvens de Internet das Coisas, no trabalho descreve que a utilização de virtualização por contêineres vem de encontro com necessidade de virtualizar dispositivos com capacidades computacionais mais limitadas, que não suportariam a sobrecarga da virtualização completa, mas que se beneficiam do fato de ter uma opção mais leve (*lightweight virtualization*). Neste cenários, os objetos inteligentes necessitam de flexibilidade para a implantação, atualização e customização do seu *software*, o trabalho discute os

benefícios de técnicas de virtualização para a performance e o gerenciamento de serviços, a experimentação foi realizada em um dispositivo embarcado *Raspberry Pi*. Importante notar é que a análise foi feita baseado na necessidade de virtualizar o sistema a partir do dispositivo, e verificar qual a sobrecarga que a camada de virtualização vai acarretar, os experimentos indicaram que é factível a sua utilização em cenários reais. Ainda em trabalho posterior Celesti (20) introduz uma arquitetura que agrega a nuvem IoT com mais serviços de *data centers*, como *backup*, tolerância a falhas, tudo isso usando contêineres para gerenciar os *firmwares* dos diversos dispositivos.

Morabito em (65) discute sobre a convergência de tecnologias como IoT, SDN (*Software Defined Network*) e como vem aumentando a utilização de soluções baseadas em contêineres para implantação de serviços em aplicações de Internet das Coisas, o trabalho trata de virtualização a partir dos dispositivos IoT na borda. Na extensão do trabalho (66), apresenta uma análise mais profunda acerca da performance e consumo de energia e dissipação de calor baseado em diferentes cargas de trabalho. O autor ainda fala sobre algumas limitações ainda encontradas, como critérios de segurança que não faziam parte do escopo do trabalho e a questão sobre a migração a quente de aplicações entre dispositivos.

A virtualização de dispositivos IoT com contêineres, bem como sua avaliação de performance é explorada pelos trabalhos de Morabito (66, 67). Nestes trabalhos é reforçada a importância da virtualização leve para a computação de borda na IoT, devido a seus requisitos básicos de oferecer uma configuração eficiente, instalação e atualização em ambientes de plataformas heterogêneas. Em (67), Morabito lista os requerimentos que os dispositivos IoT de borda necessitam que podem ser obtidos usando os recursos de virtualização, escalabilidade, multi-tenância, segurança/privacidade, latência e extensibilidade.

O CloudSim Plus é um arcabouço que permite a modelagem, simulação e experimentação de infraestruturas de computação em nuvem (93). É um simulador que os desenvolvedores podem se ater em questões específicas de design de sistema a serem investigadas, sem se preocupar com os detalhes de baixo nível relacionados a infraestruturas e serviços baseados em nuvem. O *CloudSim Plus* é uma ferramenta que pode ser utilizada para a validação de propostas de implementações de serviços em nuvem, com a vantagem de poder conduzir estudos para quem não tem acesso a grandes recursos de nuvens comerciais ou públicas.

3.3 *Testbeds* e Dados de Experimentos

Quando da impossibilidade de obtenção de dados de ambientes reais para testar os algoritmos e ideias, pode-se recorrer a simulação, geração manual ou automatizada e também a *traces* ou instâncias disponibilizadas de outros experimentos.

Em seu trabalho Armac (5) trata da simulação de ambientes inteligentes baseado na construção de um simulador de uma casa virtual, construir uma casa real para experimentação de um AmI possui um alto valor financeiro. Em certos casos, a construção de miniaturas e maquetes são a base inicial como o próprio autor afirma (71). As limitações de esses representações de AmI em miniaturas é que a estrutura física predefinida é difícil de modificar, visto que para adicionar um novo espaço (quarto, banheiro) afeta toda a estrutura do modelo. Outro problema é que não é possível utilizar esses experimentos para avaliar a escalabilidade de propostas. Estas limitações que são abordadas pelo simulador proposto eHomeSimulator, ele pode permite a flexibilidade com a possibilidade de adequação de diversos cenários, usando um editor gráfico, tanto na criação de espaços quanto na questão de disposição dos dispositivos, além da possibilidade de execução de várias instâncias para aferição de escalabilidade de projetos.

Helal (47) chama a atenção para o problema constante na área de espaços ubíquos e pervasivos que é a falta de padrões de geração de dados para testar esses ambientes, o que gera muita dificuldade no processo de desenvolvimento desses sistemas. O uso de simulação pode ajudar no projeto e construção de AmI, antes de qualquer projeto entrar em produção a simulação é capaz de fornecer uma forma de prototipação que assegure a viabilidade das propostas. A proposta do trabalho é a criação de um gerador de cenários de ambientes inteligentes com a realização de atividades monitoradas por diversos sensores, utilizado uma cadeia de Markov discreta para retratar a sequência de eventos em um determinado ambiente. O cenário é caracterizado como uma máquina de estados finita onde cada nó representa um sensor, e cada aresta direcionada indica a probabilidade de transição entre o estado atual e o próximo. Desta maneira, a técnica permite gerar comportamentos sem a necessidade da existência de um espaço físico. O modelo ainda precisa de mais validações, mas pode ser utilizado como um ponto de partida nas fases iniciais de desenvolvimento, antes de partir para cenários mais complexos.

As pesquisas na área de *Smart Home Environment* (Casas Inteligentes) já são realizadas a bastante tempo, algumas implementações reais foram realizadas e apresentaram muitas contribuições. O Instituto de Tecnologia da Geórgia criou em 1999 o primeiro

laboratório deste tipo, chamado *Aware Home* (54) a casa serve de laboratório de computação ubíqua, sua intenção é realizar o monitoramento de localização e reconhecimento de atividades humanas, equipada com vários tipos de sensores os trabalhos desenvolvidos tem o objetivo de melhorar o bem estar dos moradores, porém, devido a questões legais, nunca foi realizado um estudo com moradores por um longo período. Outro projeto, o *UMASS Intelligent Home* da Universidade de Massachusetts em Amherst utilizou sistemas multiagentes para aplicação em um AmI, através de simulações os agentes simulavam o aquecedor de água, cafeteira, temperatura, maquina de lavar e um robô (58).

Atualmente outro projeto de AmI da UMASS é o *Smart** (10), que tem o objetivo de melhorar o consumo de energia nas residências. Trata-se de um conjunto de casas reais com sensores que fornecem dados sobre consumo e geração de energia, temperatura e umidade. Os dados desses monitoramentos são disponibilizados com *traces* de vários períodos e configurações. Foram desenvolvidos vários trabalhos utilizando esses *datasets*, entre eles temos trabalhos sobre a diminuição do consumo de energia (64), gerenciamento de energias renováveis (109), monitoramento do ambiente através de medidores inteligentes (22), entre outros.

O *Adaptive House*(68), é um projeto de AmI que tem objetivo de prever os comportamentos dos moradores de uma casa através da avaliação do histórico de dados, para melhorar o bem estar dos habitantes antecipando suas necessidades. A casa laboratório tem mais de 75 sensores para os seguintes dados, temperatura, luminosidade, som, movimento, travamento de portas e janelas, e também atuadores para controlar calefação, aquecimento de água, luz e ventiladores. Projeto não lista a disponibilidade de traces, nem outra forma de acesso aos dados dos experimentos.

O *MavHome Smart Home* (106) foi um projeto de pesquisa multidisciplinar na Universidade do Estado de Washington e na Universidade do Texas em Arlington, seu foco era na criação de um ambiente doméstico inteligente. No MavHome a casa inteligente é como um agente inteligente que percebe seu ambiente através do uso de sensores e pode atuar sobre o meio ambiente através do uso de atuadores. A casa tem alguns objetivos gerais, como minimizar o custo de manter a casa e maximizar o conforto de seus habitantes. Para atingir essas metas, a casa deve ser capaz de raciocinar e adaptar-se às informações fornecidas (33). O MavHome ainda disponibiliza os *datasets* gerados em formato texto no site do projeto.

Em seu trabalho Cook (30) descreve o crescimento do interesse pelos ambientes pervasivos, porém métodos de avaliação dessas tecnologias são limitados e difíceis de serem

conduzidos. Um dos motivos é a falta de *datasets* com boa qualidade de dados, o projeto CASAS (29, 94) foi criado para avaliar vários algoritmos de reconhecimento de atividades as chamadas ADLs (Activities of Daily Living (100)), nele é possível baixar *traces* de diversos experimentos de monitoramento de AmI.

O IoT-Lab é um projeto com o objetivo de permitir a ciência aberta e a pesquisa reprodutiva, faz isso disponibilizando uma plataforma em grande escala para desenvolvedores e pesquisadores melhorarem seus produtos e protocolos. A infraestrutura do IoT-LAB consiste de um conjunto de nós em um *backbone* de rede global que fornece energia e conectividade entre todos os nós sensores, bem como, garante o sinal de rede necessário para o propósito de monitoramento da rede. Os *testbeds* são localizados em seis diferentes lugares na França, e dão acesso a 2728 dispositivos sensores sem fio, robôs móveis, entre outros. Uma grande vantagem do IoT-Lab é que o desenvolvedor tem acesso para reprogramar os dispositivos, carregar seu próprio *firmware* para analisar seu desempenho, ele é bastante utilizado para análise de protocolos de redes sem fio (2).

	Disponibiliza dados	Ativo	Acesso Aberto	Atividades Reais
IoT-Lab (2)	REST API	Sim	Sim	Sim
CASAS (29)	Traces	Sim	Não	Sim
MavHome (106)	Traces	Não	Não	Sim
Adaptive House (68)	Não	Não	Não	Sim
UMASS Smart* (10)	Traces	Sim	Não	Sim
Aware Home (54)	Não	Sim	Não	Sim

Tabela 2: Disponibilização de dados pelos *testbeds*

A Tabela 2 sintetiza que os *testbeds* de AmI disponibilizam dados dos seus sistemas para que sejam utilizados por pesquisadores externos em suas pesquisas. Modelos como o IoT-Lab, que são mais abertos e que permitem o acesso a programação dos sensores, são minoria, um problema desses ambientes de testes com acesso compartilhado é que para realizar um experimento é separado uma fatia de tempo curta, geralmente se aplicam bem para quem trabalha com avaliação de protocolos e mobilidade em RSSF. Já ambientes como o CASAS e Smart*, são bases importantes de *traces* de experimentação pois são implantados em casas convencionais, e publicam seus dados para *download* regularmente.

4 Abordagem proposta

4.1 Arquitetura

Nesta arquitetura especificamos o funcionamento da nuvem que dará suporte à implementação de aplicativos ubíquos, esses aplicativos receberão dados de Ambientes Inteligentes ou dispositivos IoT que serão colaboradores do sistema. O modelo e a especificação de aquisição de dados de IoT são descritos em trabalhos anteriores (74). Essa arquitetura pode ser usada para a construção de palavras reais e aplicativos de negócios, bem como para colaboração em pesquisa e aprendizado.

Computação ubíqua (104) tem por definição a característica de estar relacionada ao ambiente do usuário, onde vários computadores ou objetos computacionais são distribuídos em uma área física específica e a interação com os usuários ocorre de forma diferente da operação clássica de computadores, geralmente sem interfaces e botões a interação é feita com a aplicação ubíqua analisando o comportamento humano por meio de sensores e respondendo ao usuário por meio de atuadores para a realização de tarefas ou atividades importantes para o bem-estar de seus usuários.

Para atingir seu objetivo, uma aplicação onipresente precisa fazer uso e cooperar com várias outras tecnologias. Para obter dados de dispositivos e sensores, é necessário conectar-se a várias tecnologias de interconexão, como tomadas, lâmpadas, sensores de movimento e outros. Esses dados podem ser obtidos por meio de uma conexão IP, bluetooth ou serial, portanto, dada a heterogeneidade desses dispositivos e de seus procedimentos, eles fazem parte do escopo da Internet das Coisas e costumam ser utilizados middlewares para atender a esses desafios.

Um **IoT Service** possui uma representação da conexão entre sensores dentro de ambientes inteligentes, sejam eles portas, termostatos e outros, com o serviço em nuvem. Em seguida, o serviço de IoT na nuvem é responsável por registrar os sensores em seus ambientes, bem como descrever suas propriedades e recursos. No caso de adicionar um

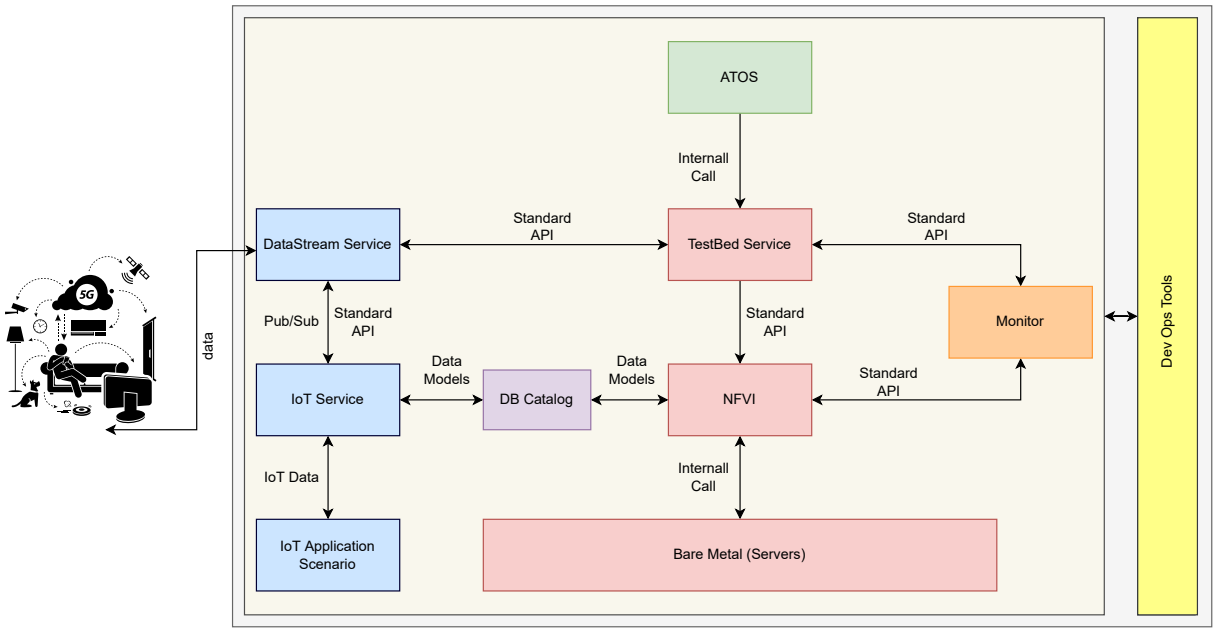


Figura 8: Arquitetura genérica da cloud com os componentes e interações.

cenário de teste, um descritor de todos os dispositivos e comandos associados deve ser enviado, para isso o Serviço IoT usa **SGR**, para mais informações consulte a seção 4.4 e ainda indicamos a leitura do artigo publicado (74).

Assim, incluímos a função SGR em nosso IoT Service para ser responsável pela integração de sensores e atuadores com a comunicação com os elementos que rodam em nossa nuvem, ou seja, os aplicativos ubíquos. O **DataStream Service** tem a função de controlar a comunicação e tratar milhares de conexões simultâneas. Para isso podemos utilizar um middleware em conjunto com a camada **IoT Service**, e a integração com outras ferramentas e aplicativos dentro da nuvem pode ser feita através da comunicação Publish/Subscriber ou API REST.

O **Dashboard GUI** é uma interface web, onde os usuários testbed podem fazer uso das funções do sistema, como a inclusão de um ambiente inteligente ou qualquer outro cenário com sensores e atuadores. É também aqui que os ambientes de teste disponíveis são listados, bem como imagens de contêiner, rastros experimentais e outros artefatos são armazenados no Repositório de Serviço. O Dasboard GUI tem interação direta com o **TestBed Service**, que é responsável por instanciar um cenário de teste e fornecer os conjuntos de software necessários, além de cuidar do gerenciamento das funções de virtualização através do **NFVI**. A instanciação se dá através do **DB Catalog** que é um *playbook* (usa descritores de arquivo como yaml e json), é através deles que descrevemos as ferramentas de software utilizadas, configurações de rede e qual função cada nó irá

desempenhar durante a execução. O Testbed Service utilizando um *playbook* do DB Catalog é capaz de lançar a execução de uma aplicação, neste contexto, todos os conjuntos de software que utiliza, como banco de dados, servidor web e a própria aplicação, fazem parte da aplicação (cada um pode ser um container), a execução desse conjunto de softwares ocorre dentro do mesmo espaço de recursos (rede, memória e processamento) é denominado POD, este recurso é realizado por algumas camadas, como NFVI e posteriormente o orquestrador da aplicação. Para a função de orquestrador, podemos utilizar tanto Kubernetes quanto Openshift ou o próprio Docker.

4.2 Decisões de Projeto

O processo de virtualização vem representando um fator importante nos processos de experimentação em Internet das Coisas, como verificamos na Seção 3.2. Considerando as experiências reportadas nos trabalhos relacionados, o desenvolvimento de experimentação em ambientes de nuvem pode se beneficiar dos seguintes atributos não funcionais que a virtualização oferece:

- **Ambiente Controlado:** Recursos definidos (CPU, Memória, Disco, Rede);
- **Bibliotecas:** Mantendo um contêiner é possível manter a bibliotecas e APIs compatíveis com a aplicação, sistema operacional e *kernel*;
- **Facilidade:** Processo de migração de um experimento para rodar na nuvem ou em uma outra máquina na rede ou local é feito de forma amigável, bem como a replicação de tarefas;
- **Escalabilidade e Elasticidade:** Utilizando contêineres, é possível criar várias instâncias de trabalhos, o custo de desempenho é mínimo comparado com outras estratégias. Permite que sejam alterados parâmetros do ambiente em tempo de execução, além de ser possível coletar as informações de execuções dos experimentos de forma mais simples e isolada das demais atividade do sistema operacional.

4.3 Modelo de Serviço

A Figura 9 apresenta o modelo básico de um serviço de nuvem para execução de experimentos em sistema ubíquos, onde o usuário entra com as configurações da experimentação em um portal Web. O sistema fornece a listagem de dados disponíveis e o usuário carrega

sua aplicação. O sistema deve ser responsável por executar a aplicação desenvolvida pelo usuário de forma isolada e respeitando os recursos disponíveis na nuvem. Já o gerenciador é responsável por criar o ambiente virtual com o *middleware*, as configurações de rede e as dependências de bibliotecas da aplicação. Ao final da execução o usuário terá disponibilizado todos os *traces* e informações sobre a execução de seu programa, bem como as informações que forem provenientes de seu próprio sistema.

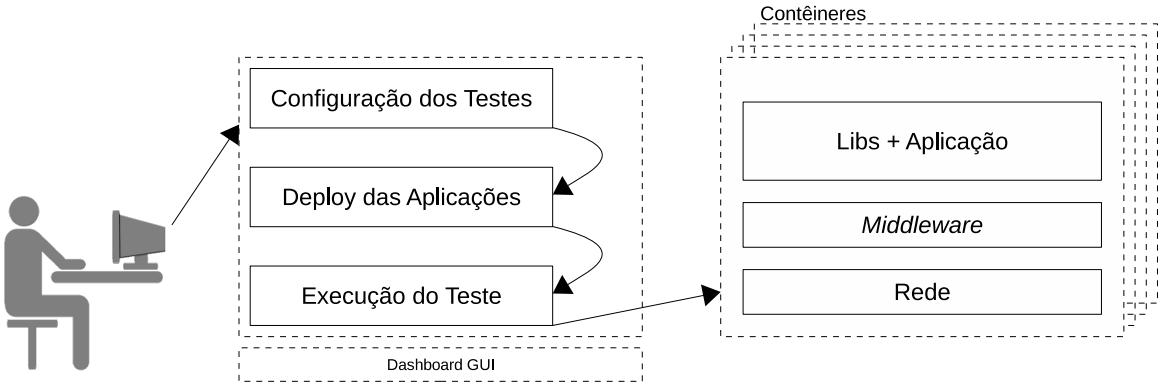


Figura 9: Modelo de Serviço de Experimentação na Nuvem

Visão geral do arcabouço de gerência de dados é apresentada no diagrama da Figura 10, ele apresenta os componentes necessários para atender os requisitos do arcabouço de gestão do dados dos dispositivos. A descrição de cada um dos componentes é mostrada logo abaixo:

- **Sensor Virtual:** O Sensor Virtual servirá de base para a conexão de dados dos sensores reais, representação da interface para comunicação com os dados gerados sinteticamente e a abstração para acesso aos traces listados no *testbed*. Outra função importante é que em um ambiente em que tenhamos recursos compartilhados a limitação de recursos dos dispositivos IoT (principalmente energia e comunicação) pode ser afetada com a quantidade de acessos ao sensor físico, desta forma, gerenciando os dados lidos e a periodicidade das leituras. O sensor virtual, como uma instância ou representação de um dispositivo, terá os dados mais recentes e as regras de leitura e envio de ações para os atuadores, por exemplo, quando vários usuários solicitam a leitura de um sensor de temperatura muitas vezes consecutivas em um curto espaço de tempo ao final respeitado a regra de periodicidade deverá ser gerada apenas uma leitura no sensor físico e as demais serão respondidas com o valor em *cache*.

- **Sensores e Atuadores:** São os principais componentes de sistemas de Internet das

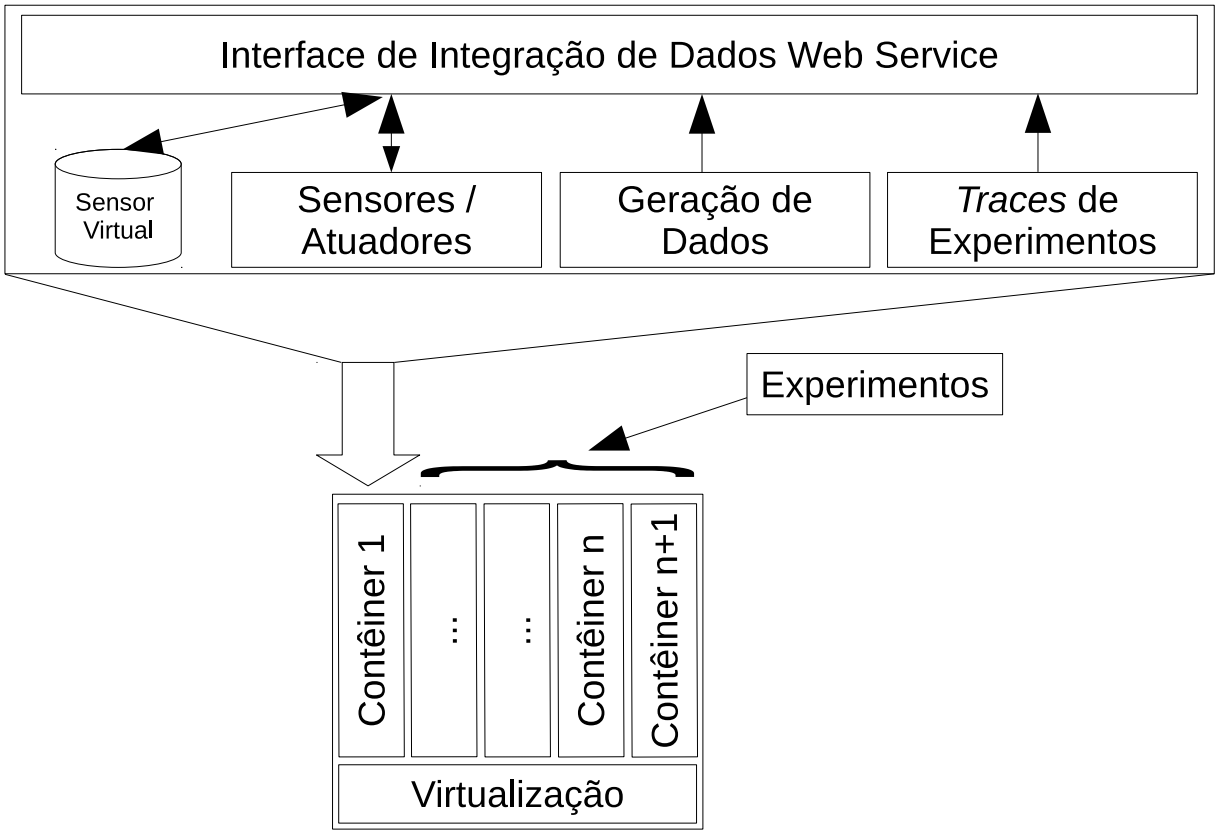


Figura 10: Visão Geral do Arcabouço de Gerência de Dados

Coisas (visto que são as "Coisas"), responsáveis por capturar os dados do mundo real e transformarem em uma informação digital, os sensores podem ser de vários tipos, nessa proposta podemos incluir desde sensores de RSSF (Redes de Sensores Sem Fio), a Arduinos e SoC (*System On Chip*). Outra parte importante é que em vários experimentos recentes os sensores também tem sido usados para processamento na borda (*IoT Edge*), nesta proposta é possível utilizar esse conceito também, tanto utilizando equipamentos reais quanto utilizando uma imagem virtualizada do dispositivo para isso. Os Atuadores são dispositivos com a capacidade de realizar uma ação ou interação com o mundo real, como acender uma luz, mudar a temperatura do ambiente, trancar uma porta entre outras coisas. Os atuadores precisam implementar regras de controle, as chamadas ACL (*Access Control List*), pois como interagem com o mundo real podem causar inconveniências aos usuários desses ambientes. Nesta proposta os sensores para participarem da rede/infraestrutura de testes, precisam anunciar seus recursos, no caso dos sensores os tipos de dados que fazem leitura e para os atuadores quais os comandos ("*ligar*", "*desligar*", "*aumentar*", "*diminuir*"e etc) podem executar, fica a cargo do Serviço de integração de dados gerir essas regras e inserir nos respectivos *sensores virtuais*.

- **Interface de Integração de Dados e *Web Service***: É uma camada com várias funções, uma delas é a interface com o usuário que é o primeiro contato do desenvolvedor com o ambiente de testes, é nessa interface que possibilita o usuário gerenciar seus testes e a obtenção dos relatórios de suas execuções. Além de fazer a interface com os usuários essa camada ainda se encarrega de fazer a interconexão entre o *middleware* de IoT, sensores virtuais e banco de dados de *traces*, para disponibilização dos dados que são usados na experimentação, esses dados devem ser disponibilizados usando tecnologias *web* tradicionais (*SOA, SOAP, API REST*), bem como a troca de mensagens padrão.

  Outra função que deve ser incorporada a esta camada é a interface de configuração do ambiente de experimentação, onde poderemos atribuir as configurações do tipo dos dados que será utilizados, bem como, os *setup* de rede para o caso de utilizar várias instâncias de contêineres.

- ***Traces* de Experimentos:** Existem muitos trabalhos em que foram construídos excelentes ambientes pervasivos, com uma boa qualidade de dados que podem ser explorados para vários outros trabalhos. Logo, a proposta tem a meta de servir de plataforma de testes onde os usuários possam escolher um tipo de experimento que mais se adéque a seus objetivos e desenvolver sua ideia sem a necessidade de montar o ambiente pervasivo, apenas reaproveitando essa parte de trabalhos já recomendados.

- **Virtualização:** A parte de virtualização é a plataforma base de todo o arcabouço, visto que o próprio sistema é executado dentro de um contêiner. Quando da execução de experimentos, estes serão executados de forma isolada dentro de um contêiner próprio. Esse modelo é uma forma de conservar testes já realizados, pois utilizando virtualização é possível gravar uma imagem do experimento, mantendo uma versão funcional do *software* e conservando todas as suas dependências para execução futura. Outro beneficio desse recurso é a possibilidade de gerar várias instâncias do experimento, para testes em que se necessite testar a escalabilidade de propostas. Além de oferecer uma característica importante no desenvolvimento de sistemas ubíquos que é o isolamento, utilizando essa camada de virtualização obtemos um ambiente com processos e endereços de memória isolados dos demais.

Podemos utilizar o modelo *SenaaS* discutido na Seção 2.3 para o compartilhamento de recursos IoT. A partir desse ponto é necessário a implementação da Gerência do Arca-

bouço, que vai possibilitar a interface com os usuários do sistema, esta Gerência também será responsável pelo controle dos usuários e gestão dos testes.

O controle de execução dos containers também apresenta alguns desafios, como o tratamento de escalabilidade de recursos, principalmente memória e processamento. Neste modelos é possível adicionar mais computadores para a formação de um *cluster* de testes, o que pode gerar mais um problema de compartilhamento de recursos e escalonamento dos contêineres. Para o modelo de virtualização proposto, será implementado um sistema de politicas de execuções de experimentos, sempre que um contêiner é criado existe a especificação dos recursos a serem utilizados, com base nessas informações o sistema só deverá permitir a criação de novas tarefas apenas quando tiver recursos suficientes disponíveis.

4.4 Camada de Serviço IoT

A camada de Serviços IoT é composta por um sistema que conecta os dispositivos IoT. Esses objetos devem se conectar a uma camada de nuvem, responsável pela virtualização de dispositivos. Esses dispositivos fornecem seus recursos para as aplicações, sejam esses sensores ou atuadores. Essa arquitetura registra o ambiente inteligente e os seus componentes e dispositivos e oferece como um serviço de rede (73).

Expor dispositivos na forma de serviço de rede pode favorecer o compartilhamento de suas informações e facilitar no processo de acesso de aplicações. A Figura 11, apresenta a camada denominada Sistema de Gerência de Recursos (SGR), ela é composta por três camadas diferentes que interagem entre si e são fundamentais para a conexão dos dispositivos IoT, com as aplicações que rodam na nuvem. Abaixo a descrição de suas camadas.

1. **Dispositivo**: é uma camada em que os dispositivos desempenham um papel importante. Eles possuem um sistema embarcado aprimorado com um ciclo de processamento em que (1) num primeiro momento se conecta e registra na camada de nuvem na primeira vez; (2) reúne dados de todos os seus sensores para serem enviados para a camada de nuvens também; (3) recebe das ações da camada Nuvem que devem ser executadas pelos atuadores do dispositivo. O dispositivo é usualmente um sistema embarcado que controla um ou mais sensores e atuadores, e em intervalos periódicos envia os dados para a *Cloud*, bem como escuta por comando para ativar seus atuadores.

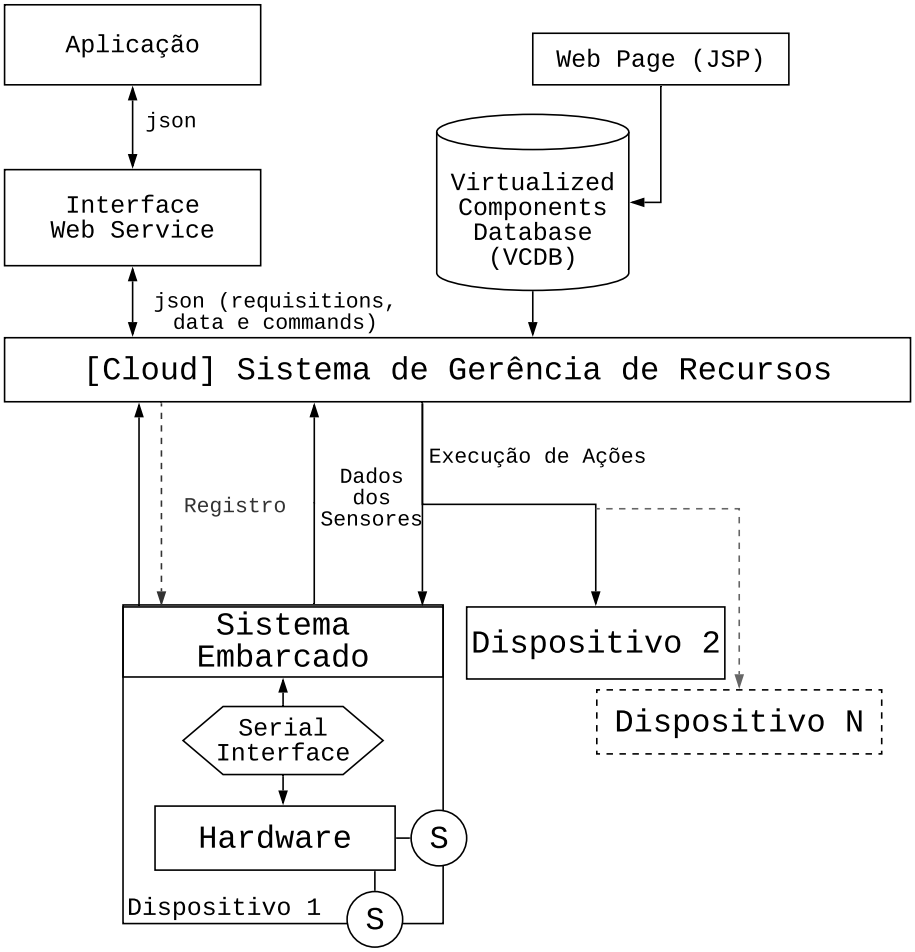


Figura 11: Os componentes do Sistema de Gerenciamento de Recursos (SGR).

2. **Nuvem**: é a camada central da arquitetura, onde há uma instância do servidor capaz de manter informações atualizadas de dispositivos hospedados em ambientes e executando em um middleware de IoT. O Sistema de Gerenciamento de Recurso (SGR) nesta camada gerencia (1) o processo de registro de dispositivos; (2) o processo de atualização de dados vindo de sensores e; (3) as ações que devem ser executadas por dispositivos. Além disso, também é capaz de expor serviços web RESTful para várias finalidades.

3. **Cliente**: é a camada responsável pela visualização dos dispositivos e utiliza diversas soluções web. Neste documento, ele é representado como uma solução da Web para visualização de ambientes e dispositivos em tempo real. Ele implementa um mecanismo de publicação e assinatura para permitir que desenvolvedores e pesquisadores obtenham informações de sensores reais, sem a necessidade de possuí-los, e oferece uma infraestrutura para aqueles que desejam fornecer sensores como um serviço.

No SGR, especificamente na camada do Dispositivo, um dispositivo é um objeto IoT

equipado com um microcontrolador (hardware) no qual sensores e atuadores são conectados. Os dispositivos hospedam um sistema embarcado capaz de se conectar e se registrar na camada de nuvem para fornecer dados atuais a serem consumidos pelos usuários. Basicamente, o sistema embarcado registra dinamicamente o dispositivo no SGR, enviando todas as suas funcionalidades (incluindo dados de recursos disponíveis e comandos de ação dos atuadores) e as informações de qual ambiente está situado. Em seguida, o dispositivo recebe uma confirmação de registro e começa a enviar dados coletados de seus sensores diretamente para o SGR, que mantém o valor mais recente disponível para ser consumido posteriormente e pode ser visualizado em um painel da Web na camada Visualização. Além disso, o sistema embarcado lida com os comandos de ação vindos do SGR que precisam ser executados pelos atuadores do dispositivo.

O SGR é o principal componente da Camada **Iot Service** e é responsável por manter informações sobre dispositivos e ambientes a serem consumidos como um serviço por outras camadas. Possui um mecanismo para registrar novos dispositivos em determinados ambientes, armazenando suas informações no Banco de Dados de Componentes Virtualizados (VCDB). O VCDB também mantém atualizados os dados provenientes de dispositivos, considerando apenas os novos valores que foram alterados desde a última mensagem de dados recebida. Além disso, essa camada também expõe os serviços da Web para fornecer diferentes serviços, como camadas de visualização para aplicativos móveis, permitindo que novas tecnologias interajam e acessem o VCDB e, consequentemente, expandam as funcionalidades do SGR. Os serviços da Web são expostos em uma arquitetura RESTful para interoperabilidade do sistema e usam arquivos Json para trocar informações (dados, requisições, comandos de execução, etc.) entre as camadas Nuvem e Cliente.

As ações que precisam ser executadas pelos sistema embarcado na camada de dispositivo são gerenciadas pelo SGR. Quaisquer usuários ou serviços que desejem usar ou executar uma ação em qualquer atuador disponível devem enviar um comando de ação acessando um dos serviços fornecidos. O SGR redireciona a ação diretamente para o dispositivo específico, que é capaz de executar ações no hardware. As informações técnicas sobre hardware e atuadores são transparentes para o usuário final (e gerenciadas por essa camada), que precisam apenas conhecer os comandos disponíveis do dispositivo que ele deseja interagir. Enquanto um dispositivo está sendo usado por alguém, o SGR pode bloquear o atuador específico ou o dispositivo inteiro para evitar conflitos de execução na camada inferior. O respectivo recurso é desbloqueado após a execução ou após um limite de tempo limite.

O SGR é uma solução do lado do servidor que executa ao lado de um *gateway* de um *middleware* IoT na qual os dispositivos devem se conectar. O middleware deve fornecer a conectividade e a capacidade de comunicação necessárias para que os dispositivos interajam com a camada do SGR. Além disso, o middleware escolhido deve garantir a escalabilidade do sistema. Mais detalhes técnicos sobre o middleware escolhido são fornecidos mais adiante na seção 4.4.1.

A camada Cliente é responsável por disponibilizar interfaces aos usuários finais por meio do uso de soluções da Web, como serviços da Web e páginas, para permitir que eles consumam os serviços virtualizados fornecidos pela camada de nuvem. Na verdade, os usuários são os principais consumidores dos dispositivos disponíveis na camada de nuvem, no entanto, outros aplicativos inteligentes (como programas ou algoritmos de agentes) podem acessar a camada de nuvem se um serviço da Web direcionado for exposto a eles. Para facilitar o gerenciamento de ambientes, essa camada do Cliente fornece uma página da web na qual todos os ambientes podem ser acessados em tempo real. Esta página web é um sistema onde o usuário pode criar seus próprios ambientes definindo algumas informações básicas e se é um ambiente público ou privado por exemplo. Este sistema da web mostra todos os ambientes públicos que permitem aos usuários selecionar e ver suas informações disponíveis. As pessoas devem acessar o painel web e selecionar os sensores que desejam se inscrever para ter acesso a um grupo específico e personalizado de informações. Neste trabalho, um ambiente é definido por uma representação lógica de um espaço físico onde vários dispositivos podem coexistir compartilhando informações em uma rede IoT.

4.4.1 Implementação das Camadas

Nesta seção, é apresentada a implementação das camadas Dispositivo, Nuvem e Cliente considerando componentes tecnológicos e como essas camadas se comunicam entre si. Para as camadas Dispositivo e Nuvem, empregamos o middleware ContextNet (40), que é um middleware IoT para compartilhamento de dados em um ambiente de grande escala.

O ContexNet é um serviço de fornecimento de contexto para redes fixas e móveis, que possui uma camada chamada SDDL (97) e é usado para comunicação de dados. Esse serviço já aborda os principais problemas de comunicação de dados, como tolerância a falhas, balanceamento de carga, suporte a desconexão de nó (Handover) e segurança. Ele também possui recursos dinâmicos de criação e gerenciamento de grupos, além de usar o modelo de comunicação publish/subscribe.

No ContextNet, a transferência de dados é feita através de dois protocolos, o MR-UDP (92) e o OMG DDS (78). O MR-UDP manipula mensagens entre o cliente e um documento de texto, uma vez que o DDS está na distribuição de dados no núcleo da rede. Com o uso do ContextNet é possível possibilitar o crescimento da rede, garantindo uma escalabilidade da distribuição dos materiais.

Além disso, a camada Dispositivo usa uma interface serial de baixo acoplamento (56) para comunicação com os microcontroladores responsáveis pela hospedagem dos sensores e atuadores. Essa interface isola o alto nível da programação de baixo nível, usando comandos seriais para ativar as funções pré-definidas dos usuários, que respondem executando ações em atuadores ou reunindo dados de sensores. A camada Dispositivo emprega um sistema web simples capaz de gerenciar ambientes, registrando novos e mostrando seus recursos disponíveis para os existentes, e é aprimorado com um mecanismo de publicação e assinatura para usuários interessados em recursos específicos. Apesar do uso de um sistema da Web neste documento, toda a abordagem permite expor serviços da Web para explorar outras soluções da Web e móveis.

A SGR expõe sensores e atuadores como um serviço a ser consumido por terceiros, usando soluções web. Todos os componentes do SGR consideram um modelo em que esses recursos são acoplados em dispositivos situados em ambientes pré-existentes no sistema. Por exemplo, no estado atual do sistema os ambientes devem ser registrados manualmente pelo usuário para facilitar o controle dos ambientes, disponíveis e para o controle de segurança.

4.4.2 A Camada Dispositivo

A camada *Dispositivo* é um sistema embarcado em execução em um dispositivo com capacidade de processamento suficiente e capaz de fazer interface com sensores e atuadores. Há uma arquitetura física e lógica. A arquitetura física compreende tecnologias de hardware e é composta de uma pequena placa de computador móvel com conexões *Bluetooth* e Wi-Fi (por exemplo, Raspberry Pi) conectada a um ou mais micro-controladores usando comunicação serial. Os microcontroladores são responsáveis por controlar os sensores e os atuadores em baixo nível. A arquitetura lógica da camada *Dispositivo* compreende tanto a programação dos microcontroladores quanto o sistema embarcado. Os microcontroladores são capazes de coletar os dados brutos dos sensores e enviá-los para o sistema embarcado hospedado no minúsculo computador. Depois disso, o sistema embarcado envia os dados para a camada *Nuvem* ou recebe comandos para serem enviados ao microcontrolador, a

fim de executar ações usando os atuadores.

O micro-controlador é programado em *loop* para verificar se há mensagens vindas do sistema embarcado para coletar informações ou executar ações. Portanto, se a ação de coleta for ativada, ela acessa todos os sensores e monta uma *string* a ser enviada pela comunicação serial para o sistema embarcado. Caso contrário, a ação de execução é verificada e se existir um equivalente no microcontrolador, uma ação é executada no respectivo atuador.

O sistema embarcado é responsável pelo controle do microcontrolador e é um cliente ContextNet capaz de se comunicar com a camada Nuvem. Para isso, há também uma estrutura de dados para sincronizar a recepção dos dados provenientes dos sensores a serem enviados para a camada Nuvem e as ações que precisam ser enviadas e executadas no microcontrolador, porque ambos não podem executar ao mesmo tempo para evitar conflitos indesejados. Assim, as ações recebidas da camada Nuvem são colocadas em uma fila de ações para serem executadas uma etapa após os dados dos sensores serem coletados através de portas seriais.

O sistema embarcado é capaz de realizar uma autoconfiguração e registro na camada de nuvem quando ele começa a ser executado e há um servidor de nuvem disponível. Para isso, o dispositivo mantém um arquivo XML descrevendo todos os recursos disponíveis, comandos e as informações de configuração do servidor. Por exemplo, esse arquivo deve ser fornecido pelo designer no tempo de design. No sistema embarcado, o XML é a única interação necessária entre o sistema e o designer. Depois que o arquivo é preenchido corretamente, o sistema executa tudo automaticamente. As seguintes informações devem ser fornecidas:

1. **Servidor**: o *gateway* do servidor e a porta devem ser fornecidos para se conectar à camada de nuvem na qual a instância do servidor do ContextNet está instalada. Além disso, o intervalo de tempo de envio de dados para o servidor deve ser definido.

2. **Environment**: por exemplo, é necessário informar em qual ambiente o dispositivo é inserido. Para isso, o número de identificação do ambiente deve ser declarado no arquivo de configuração. A identificação do ambiente é gerada quando os usuários registram um ambiente no sistema da web.

3. **Resources**: todos os recursos disponíveis devem ser mapeados no arquivo de configuração. Todos os recursos possuem a porta serial onde estão conectados, seu nome

e uma descrição não obrigatória. Se o recurso for um atuador, ele também possui os comandos de execução disponíveis.

4.4.3 A Camada de Nuvem

A camada de nuvem é uma instância do servidor do ContextNet que executa um sistema central responsável pela virtualização de ambientes, dispositivos e recursos disponíveis. Os dispositivos são capazes de se registrar na nuvem informando seus recursos e ambiente onde estão situados. O processo de registro começa quando a solução principal recebe um objeto contendo as informações do dispositivo que deseja ser registrado. Em seguida, o sistema extrai as informações do objeto e registra o dispositivo no sistema no ambiente informado. Depois, envia de volta uma mensagem para a instância dos dispositivos do ContextNet autorizando o início da recepção de dados dos sensores.

Uma vez que os dispositivos começam a enviar os dados, o sistema central registra os novos valores e atualiza os antigos em um banco de dados (o VCDB, veja a Figura 11). Para cada dispositivo, existem recursos que podem ser sensores ou atuadores. No caso de sensores, se for a primeira vez que um novo valor de um recurso é recebido, o sistema insere esse novo valor no banco de dados. Caso contrário, se o valor tiver sido alterado desde a última recepção de dados, esse novo valor será atualizado no banco de dados. No caso de atuadores, não há dados a serem armazenados, mas é mantida a informação da disponibilidade do recurso. Por exemplo, o sistema informa se um determinado atuador está sendo usado ou está livre para executar comandos.

Como dito anteriormente, o sistema central lida com requisições de comandos a serem enviadas e executadas em dispositivos. As requisições vêm da camada Cliente (um sistema da web ou por serviços da Web expostos na camada de nuvem). A camada Cliente não precisa conhecer detalhes técnicos do hardware ou até mesmo em qual dispositivo o comando será executado. Os comandos são específicos para um recurso e o sistema principal gerencia evitando recursos e comandos duplicados. Além disso, ele também redireciona um comando recebido para o respectivo dispositivo registrado no sistema enviando uma mensagem para a instância dos dispositivos do ContextNet. É importante observar que cada dispositivo informa os comandos de seus recursos quando se registra na camada de nuvem.

Na camada Nuvem, há também um processo de bloqueio para evitar conflitos na execução de ações quando dois ou mais clientes tentam executar comandos no mesmo atuador. Para isso, toda vez que um cliente precisar usar um atuador específico, o sistema

central bloqueará esse recurso até que o cliente informe que não está mais usando, a guerra de ação executada ou um limite de tempo limite atingido. Durante esse processo, o recurso bloqueado fica indisponível para todos os outros clientes. É importante observar que os sensores não fazem parte do processo de bloqueio porque a natureza dos sensores neste documento é fornecer dados públicos a serem consumidos pela camada do Cliente. Se os dados forem considerados confidenciais, o ambiente pode ser definido como privado.

Como mencionado anteriormente, os serviços da Web podem ser expostos para estender as funcionalidades do SGR na camada de nuvem. Para criar serviços da web RESTful, é usado um contêiner de servlet incorporado e um servidor de aplicações Web. Os serviços da Web disponíveis são: um serviço de execução para ativar ou desativar os atuadores com base nos comandos disponíveis; e um serviço da Web para fornecer acesso a dados para aplicativos móveis. A Tabela 3 apresenta os protocolos que serão disponíveis para serem consumidos pelos clientes, atualmente apenas RESTful esta sendo suportado.

Protocolo	Transporte	Escalável	Segurança
REST API/tcp	Cliente/Servidor	Limitado	HTTPS
CoAP/udp	Cliente/Servidor	Sim	DTLS
MQTT/tpc-udp	Publish/Subscribe	Sim	SSL

Tabela 3: Tipos de serviços de mensagens

4.4.4 A Camada do Cliente

A ideia por trás do modelo de visualização é fornecer uma camada capaz de mostrar os recursos dos ambientes e seu comportamento em qualquer tipo de plataforma, como páginas da Web, serviços da Web ou aplicativos móveis. Além disso, é responsável por fornecer alguns mecanismos básicos que não estão disponíveis nas camadas anteriores, como criação de ambientes e mecanismos de publicação e assinatura, considerando recursos, por exemplo.

Neste estado do trabalho, a camada Cliente é representada como uma página da web para mostrar todos os recursos dos ambientes e algumas funções básicas para interagir com os recursos. O usuário é capaz de criar ambientes para hospedar virtualmente seus dispositivos e expor seus sensores como um serviço a ser consumido por outros usuários. Além disso, alguns dos atuadores têm comandos que os usuários podem ser ativados interagindo com o sistema da web da camada do Cliente. As tecnologias por trás da página da Web são bancos de dados relacionais, páginas da Web em Java e Ajax (79).

Além disso, os usuários podem optar por seguir alguns dos recursos sem a necessidade

de acessar os ambientes toda vez que precisar obter esses valores. O mecanismo de publicação e assinatura permite que os usuários tenham acesso aos valores sempre que uma mudança é percebida pelo núcleo veiculado na camada de nuvem. Além disso, o modelo de publicação e assinante permite ao usuário definir algumas regras básicas, como definir um valor desejado a ser anunciado quando atingido. Todos os módulos de visualização existentes podem coexistir sem interferir um no outro, pois todas as funções são gerenciadas pela camada de nuvem. Assim, todos eles precisam se conectar ao banco de dados da camada de nuvem ou a um serviço da Web exposto para acessar as informações desejadas.

5 Validação e Experimentos

Neste capítulo vamos apresentar o que foi construído com o intuito de avaliar a proposta, tanto baseado em simulações de *deploy* das aplicações em nuvem, quanto a implementação de uma prova de conceito. As simulações utilizando o simulador CloudSim Plus foram baseadas nos modelos modelados nas seções a seguir.

5.1 Validação da Arquitetura

De modo a otimizar o processamento das cargas de trabalho para a arquitetura proposta, dois modelos matemáticos distintos foram desenvolvidos. Em geral, o problema de alocação ótima de recursos na nuvem é frequentemente tratado na literatura a partir de modelagem matemática. Sob esta perspectiva, BARSHAN (11) propuseram um modelo de programação linear inteira mista para resolver o problema de alocação de recursos de forma centralizada e hierárquica. Já Jing XU (105), faz uma proposta de um modelo matemático capaz de reduzir, de forma eficiente, o desperdício total de recursos, consumo de energia e custos de dissipação térmica simultaneamente. Outros trabalhos correlatos fazem uso de modelos de energia linear de modo a estimar o consumo energético de clusters (60, 39).

Por sua vez, para o cenário examinado neste estudo, as cargas de trabalho ou cloudlets são implantados em containers alocados em hosts com características distintas ou não, presentes em um mesmo data center. De forma adicional, vale ressaltar que as vCPUs dos containers são homogêneas independentemente do host alocado, i.e., possuem a mesma capacidade de processamento em MIPS - Million Instructions Per Second.

Em particular, um dos modelos matemáticos compreende a alocação de containers aos hosts; em contrapartida, o outro modelo tem por finalidade reduzir o tempo total de implantação dos cloudlets. Com o intuito de analisar as abordagens mencionadas, nesta seção serão introduzidas as notações utilizadas para ambas as modelagens, assim como os modelos serão validados a partir de casos práticos.

5.1.1 Notação Matemática

Um conjunto substancial de variáveis e constantes são compartilhadas entre os modelos. Por este ângulo, a seguinte padronização será adotada. Conjuntos e variáveis serão identificadas através de letras maiúsculas, assim como parâmetros serão representados por letras minúsculas ou gregas. A partir disto, a seguinte notação será usada para a compreensão dos modelos:

$N:$	$j \in \{1..n\}$	Número de cloudlets (jobs)
$M:$	$i \in \{1..m\}$	Número de containers (machines)
$H:$	$l \in \{1..h\}$	Número de hosts
$gmips$		MIPS por vCPU
$mcpu$	$\forall i \in \{1..m\}$	Número de vCPUs no container
$hcpu$	$\forall l \in \{1..h\}$	Número de vCPUs no host
$mram$	$\forall i \in \{1..m\}$	Capacidade de Memória em GBytes no container
$hram$	$\forall l \in \{1..h\}$	Capacidade de Memória em GBytes no host
$mband$	$\forall i \in \{1..m\}$	Capacidade de Banda no container
$hband$	$\forall l \in \{1..h\}$	Capacidade de Banda no host
$ncpu$	$\forall j \in \{1..j\}$	Número de vCPUs requeridas pelo cloudlet

Ademais, é importante frisar que apesar da possibilidade de se fixar valores heterogêneos para parâmetros de hosts e containers, de forma a simplificar e viabilizar os testes práticos, foram adotados apenas valores homogêneos. Desta maneira, de acordo com o cenário apresentado outrora, os parâmetros supracitados serão definidos da seguinte forma:

$m =$	5	Número de containers
$n =$	25	Número de cloudlets
$h =$	5	Número de hosts
$gmips =$	1000	MIPS por vCPU
$nmips_j =$	10000	Demanda MIPS do cloudlet
$mcpu_i =$	4	Número de vCPUs no container
$hcpu_l =$	8	Número de vCPUs no host
$mram_i =$	2	Capacidade de Memória em GBytes no container
$hram_l =$	20(?)	Capacidade de Memória em GBytes no host
$mband_i =$	(?)	Capacidade de Banda no container
$hband_l =$	(?)	Capacidade de Banda no host

Outras notações matemáticas pertinentes ao entendimento dos modelos matemáticos, serão apresentadas, de maneira particular, a seguir.

5.1.2 Problema de Alocação de *Containers* a *Hosts*

A alocação eficiente de containers (ou máquinas) aos hosts é essencial para se obter um desempenho satisfatório na simulação. Neste sentido, foi proposto um modelo de programação linear inteira mista com a finalidade de minimizar o tempo necessário de implantação dos cloudlets, a partir de critérios de penalização para o uso excessivo de recursos computacionais em containers e hosts.

Em particular, no modelo proposto é possível exceder as capacidades físicas da CPU, memória e largura de banda através do uso de fatores de excesso, α, β e γ, na devida ordem. Contudo, a penalização resulta em uma redução significativa de desempenho, o que impacta na qualidade de serviço e no preço final ofertado. Por fim, para que sejam respeitados os limites físicos, os fatores de excesso devem ser fixados em 1. A partir das informações mostradas, os seguintes parâmetros específicos são introduzidos para o modelo em questão.

$\hat{\alpha_i}$: Fator de penalização de tempo por CPU em excesso no container

$\hat{\beta_i}$: Fator de penalização de tempo por RAM em excesso no container

$\hat{\gamma_i}$: Fator de penalização de tempo por Banda em excesso no container

α_l : Fator de excesso de CPU para host

β_l : Fator de excesso de RAM para host

γ_l : Fator de excesso de Banda para host

De modo complementar, são denotadas as variáveis X e Y.

X : $j \in \{1..n\}, i \in \{1..m\}$ aloca cloudlet em container

Z : $i \in \{1..m\}, l \in \{1..h\}$ aloca container em host

$$\min \sum_{i}^{m} \hat{\alpha_i}\ E\_cpu_i + \sum_{i}^{m} \hat{\beta_i}\ E\_ram_i + \sum_{i}^{m} \hat{\gamma_i} E\_band_i \tag{5.1}$$

$$\sum_{l}^{h} Z_{il} = 1 \quad \forall i \in \{1..m\} \tag{5.2}$$

$$\sum_{i}^{m} X_{ji} = 1 \quad \forall j \in \{1..n\} \tag{5.3}$$

$$\sum_{j}^{n} ncpu_j X_{ji} - mcpu_i \leq E\_cpu_i \quad \forall i \in \{1..m\} \tag{5.4}$$

$$\sum_{j}^{n} nram_j X_{ji} - mram_i \leq E\_ram_i \quad \forall i \in \{1..m\} \tag{5.5}$$

$$\sum_{j}^{n} nband_j X_{ji} - mband_i \leq E\_band_i \quad \forall i \in \{1..m\} \tag{5.6}$$

$$\sum_{i}^{m} ncpu_j Z_{il} \leq \alpha_l \; hcpu_l \qquad \forall l \in \{1..h\} \tag{5.7}$$

$$\sum_{i}^{m} nram_j Z_{il} \leq \beta_l \; hram_l \qquad \forall l \in \{1..h\} \tag{5.8}$$

$$\sum_{i}^{m} nband_j Z_{il} \leq \gamma_l \; hband_l \qquad \forall l \in \{1..h\} \tag{5.9}$$

$$Z_{il} \in \{0,1\} \qquad \forall i \in \{1..m\}, l \in \{1..h\} \tag{5.10}$$

$$X_{ji} \in \{0,1\} \qquad \forall j \in \{1..n\}, i \in \{1..m\} \tag{5.11}$$

$$E\_cpu_i \geq 0 \qquad \forall i \in \{1..m\} \tag{5.12}$$

$$E\_ram_i \geq 0 \qquad \forall i \in \{1..m\} \tag{5.13}$$

$$E\_band_i \geq 0 \qquad \forall i \in \{1..m\} \tag{5.14}$$

O modelo apresentado tem por objetivo minimizar o custo de implantação dos cloudlets nos containers, por intermédio de penalizações por excesso de uso de recursos computacionais. As Restrições (5.2) alocam cada container em exatamente um host. De forma similar, as Restrições (5.3) alocam cada cloudlet em um único container. Ademais, as Restrições (5.4), (5.5) e (5.6) delimitam, na seguinte ordem, a CPU, memória e largura de banda dos containers para todos cloudlets alocados, considerando os fatores de penalização. De modo análogo, as Restrições (5.7), (5.8) e (5.9) determinam o limite de CPU, memória e largura de banda dos hosts para todos os containers alocados, considerando os devidos fatores de penalização. Por fim, as Restrições (5.10) e (5.11) definem X e Z como variáveis de decisão binárias, enquanto Restrições (5.12)-(5.14) definem as variáveis como contínuas e não-negativas.

De maneira a exemplificar na prática o modelo de alocação de containers a hosts, um exemplo é conduzido. Para este cenário, o fator α_j é fixado em 50%, o que indica uma subutilização do container em relação ao host. Sendo assim, o ideal é utilizar uma penalização $\hat{\alpha}_i$ alta, com a finalidade de incentivar o balanceamento dos cloudlets entre os diversos containers existentes. No mais, para um exemplo com 5 hosts, 5 containers e 25 cloudlets, uma possível distribuição é apresentada na Tabela 4, o que, por sua vez, resultaria em um custo de CPU de 30 unidades.

host	$h_1(8)$	$h_2(8)$	$h_3(8)$	$h_4(8)$	$h_5(8)$
container	$m_1(4)$	$m_2(4)$	$m_3(4)$	$m_4(4)$	$m_5(4)$
cloudlet	$n_1(2)$	n_2	n_3	n_4	n_5
cloudlet	$n_6(2)$	n_7	n_8	n_9	n_{10}
cloudlet	$n_{11}(2)$	n_{12}	n_{13}	n_{14}	n_{15}
cloudlet	$n_{16}(2)$	n_{17}	n_{18}	n_{19}	n_{20}
cloudlet	$n_{21}(2)$	n_{22}	n_{23}	n_{24}	n_{25}
cloudlet	$n_{21}(2)$	n_{22}	n_{23}	n_{24}	n_{25}
E_cpu	10-4=6	6	6	6	6

Tabela 4: Instância de alocação em *hosts*.

5.1.3 Problema de Minimização de Tempo na Implantação de *cloudlets*

O modelo vinculado tem como objetivo principal a redução de tempo de implantação de cloudlets, com foco no tempo máximo de término do processo em $T = \{0..t_{max}\}$ unidades de tempo (tipicamente, segundos). A modelagem, de grande complexidade, considera demandas de execução MIPS por parte dos cloudlets. Por fim, as seguintes variáveis são introduzidas:

$$Exec: \quad j \in \{1..n\}, i \in \{1..m\}, t \in \{0..t_{max}\} \tag{5.15}$$

$$MHot: \quad i \in \{1..m\}, t \in \{0..t_{max}\} \tag{5.16}$$

$$MHHot: \quad i \in \{1..m\}, l \in \{1..h\}, t \in \{0..t_{max}\} \tag{5.17}$$

$$XMIPS: \quad j \in \{1..n\}, t \in \{0..t_{max}\} \tag{5.18}$$

$$HFlow: \quad i \in \{1..m\}, t \in \{0..t_{max}\} \tag{5.19}$$

$$GFlow \quad \text{contagem global de tempo para todos containers} \tag{5.20}$$

A variável Exec (5.15) é responsável pela execução da simulação, e define que o *cloudlet* j é executado no container i no tempo t. Já no MHOT (5.16) o container i é utilizado no tempo t. O MHHot (5.17) define que o container i é utilizado no host l no tempo t. O XMIPS (5.18) é a variável que calcula o mips acumulado do *cloudlet* j no tempo t. O HFlow (5.19) marca o uso acumulado do *container* i no tempo t.

Segue o modelo.

$$\min\ GFlow \tag{5.21}$$

$$GFlow \geq \sum_{i}^{m} HFlow_{i,t_{max}} \tag{5.22}$$

$$\sum_{j}^{n} Exec_{jit}\ ncpu_j \leq mcpu_i \quad \forall i \in \{1..m\}, t \in \{1..t_{max}\} \tag{5.23}$$

$$Exec_{jit} \leq X_{ji} \forall i \in \{1..m\}, j \in \{1..n\}, \forall t \in \{1..t_{max}\} \tag{5.24}$$

$$HFlow_{i,t=0} = 0 \quad \forall i \in \{1..m\} \tag{5.25}$$

$$HFlow_{i,t+1} \geq MHot_{it} + HFlow_{it} \quad \forall t \in \{1..t_{max}-1\}, i \in \{1..m\} \tag{5.26}$$

$$n\ MHot_{i,t} \geq \sum_{j}^{n} Exec_{jit} \quad \forall t \in \{1..t_{max}\}, i \in \{1..m\} \tag{5.27}$$

$$\sum_{i}^{m} MHHot_{ilt}\ mcpu_i \leq hcpu_l \quad \forall t \in \{1..t_{max}\}, l \in \{1..h\} \tag{5.28}$$

$$2\ MHHot_{ilt} \geq (Z_{il} + MHot_{it} - 1) \quad \forall t \in \{1..t_{max}\}, i \in \{1..m\}, l \in \{1..h\} \tag{5.29}$$

$$XMIPS_{j,t=0} = 0 \forall t \in \{1..t_{max}\}, j \in \{1..n\} \tag{5.30}$$

$$XMIPS_{j,t+1} \leq XMIPS_{jt} + \sum_{i}^{m} Exec_{jit}\ gmips\ ncpu_j \forall t \in \{1..t_{max}-1\}, j \in \{1..n\} \tag{5.31}$$

$$XMIPS_{j,t_{max}} \geq nmips_j \quad \forall t \in \{1..t_{max}\}, j \in \{1..n\} \tag{5.32}$$

$$XMIPSjt \geq 0 \quad \forall j \in \{1..n\}, t \in \{0..t_{max}\} \tag{5.33}$$

$$MHot_{it} \in \{0,1\} \quad \forall i \in \{1..m\}, t \in \{0..t_{max}\} \tag{5.34}$$

$$MHHot_{ilt} \in \{0,1\} \forall i \in \{1..m\}, l \in \{1..h\}, t \in \{0..t_{max}\} \tag{5.35}$$

$$Exec_{jit} \in \{0,1\} \forall i \in \{1..m\}, l \in \{1..h\}, t \in \{0..t_{max}\} \tag{5.36}$$

A Função Objetivo (5.21) tem por finalidade minimizar o *makespan* (maior atraso) contabilizado por GFlow. Por este ângulo, a Restrição (5.22) é responsável por computar o GFlow a partir dos HFlow de todos containers. As Restrições (5.23) garantem que o

consumo de CPU por um cloudlet sinalize o container com a tag "em execução". Já as Restrições (5.24) limitam execução em containers com alocação prévia a partir da fixação de valores para as variáveis X pelo modelo de alocação de containers a hosts. As Restrições (5.25) zeram o fluxo de tempo HFlow para cada container, no tempo zero. As Restrições (5.26) indicam um avanço de fluxo no tempo. As Restrições (5.27) obrigam a marcação de MHot caso exista execução (Exec) no container naquele instante de tempo. Por sua vez, as Restrições (5.28) restringem o máximo de cpu no hot, através das marcações MHHot de containers. Já as Restrições (5.29) conectam MHHot a MHot, emulando uma multiplicação quadrática $Z_{il} \cdot MHot_{it}$ em forma linearizada. Ademais, as Restrições (5.30) marcam XMIPS como zero, no tempo zero, para todos cloudlets. As Restrições (5.31) avançam XMIPS no tempo, adicionando o consumo $gmips \cdot ncpu_j$, para o cloudlet j, a cada instante de tempo. As Restrições (5.32) garantem que $nmips$ são processados no tempo final. Finalmente, As Restrições (5.34)-(5.36) definem MHot, MHHot e Exec como variáveis de decisão binárias, enquanto as Restrições (5.33) definem XMIPS como variáveis contínuas não-negativas.

5.2 Simulação

Os experimentos foram realizados no simulador CloudSim Plus (93). O objetivo é impor um ambiente que avalie a escalabilidade da arquitetura proposta, bem como obter métricas de execução considerando a modelagem desenvolvida, e o tempo total de implantação dos *cloudlets*.

5.2.1 Cenário 1

No primeiro cenário pretendemos verificar o comportamento da arquitetura em relação à atividade de implantação de diversos experimentos e medir como seria o desempenho de nossa proposta. Portanto, colocamos 25 tarefas para implantar em um data center, com 5 hosts, outros valores de configuração podem ser vistos na Tabela 5.

A Tabela 6 mostra onde os *Cloudlets* e seus respectivos contêineres foram instanciados. Na simulação, a arquitetura proposta completou a implantação dos aplicativos em 25 segundos.

As Figuras de 12 a 16 mostram o desempenho dos 5 containers durante o processo de implantação, que podemos entender como cada um sendo a execução de uma tarefa. Todos possuem um desempenho muito semelhante, finalizando suas execuções em 25

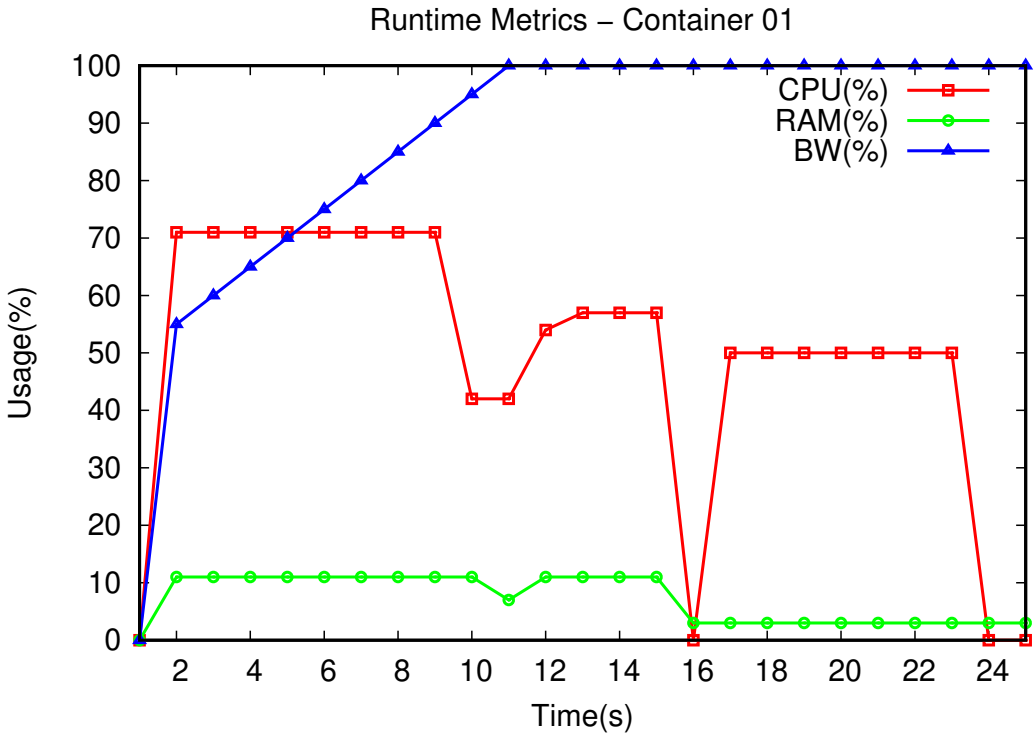


Figura 12: Implantação e execução do Container 1, no cenário 1.

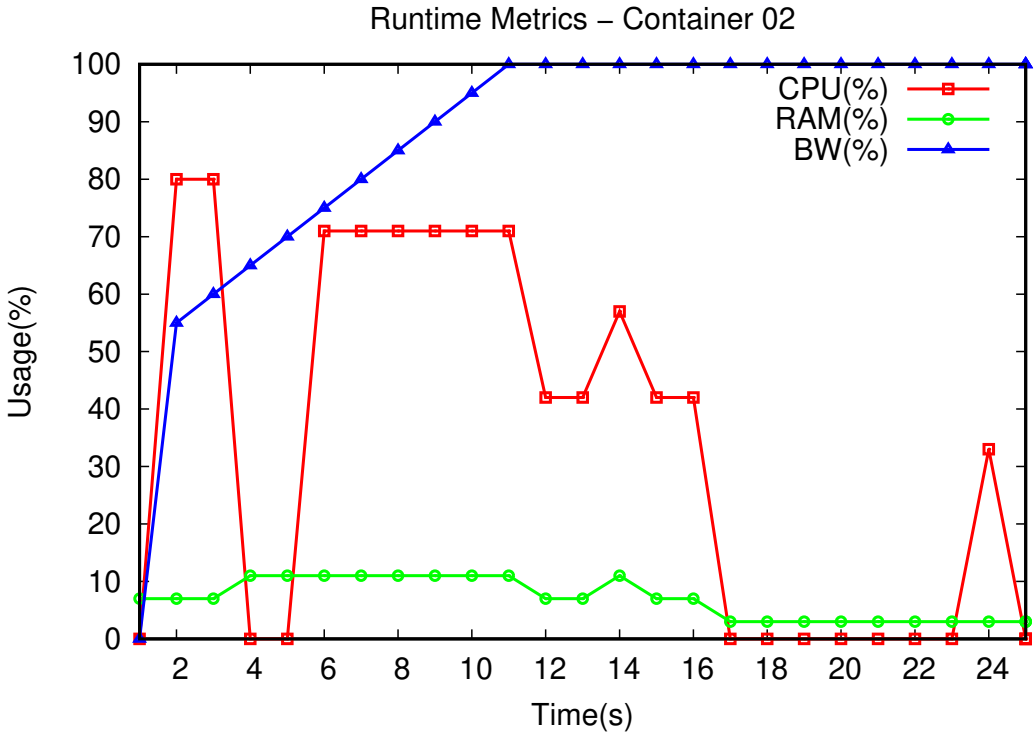


Figura 13: Implantação e execução do Container, no cenário 2.

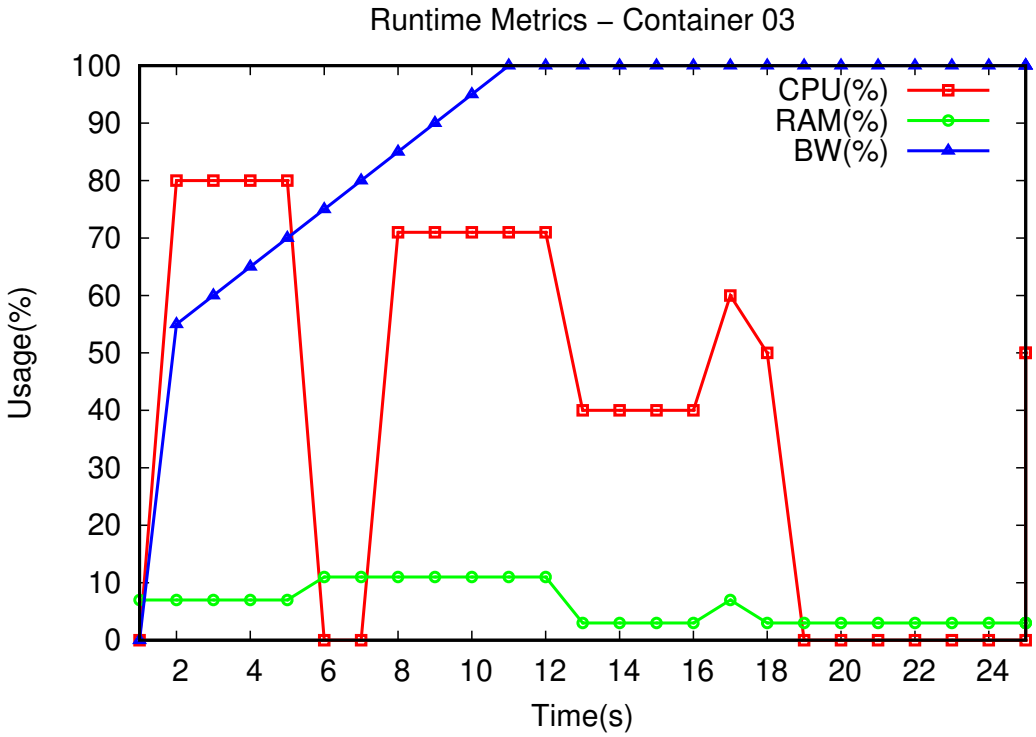


Figura 14: Implantação e execução do Container, no cenário 3.

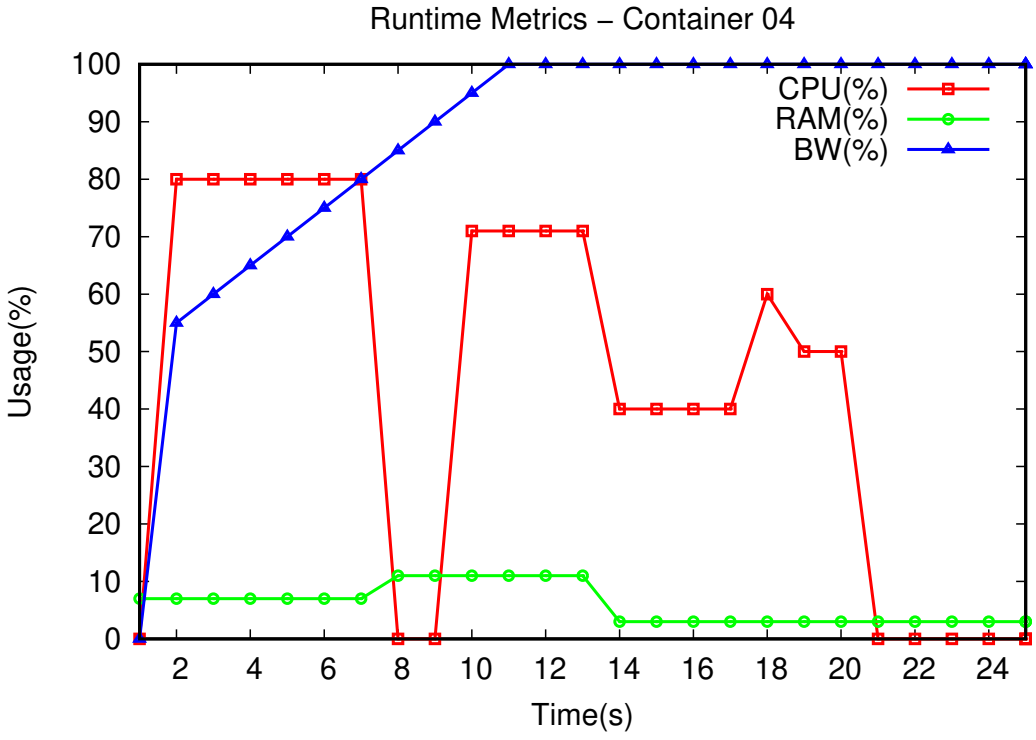


Figura 15: Implantação e execução do Container, no cenário 4.

Description	Values
Datacenter (Pod)	01
Hosts	05 with 08 vCPU
Container	05 with 04 vCPU reserved and 2048MB of RAM
Cloudlets (Queue)	25 with restriction to use 02 vCPU
Cloudlets (Instructions)	10.000 MIPS
Network Interface	10.000 Mbps

Tabela 5: Configuração da simulação - Cenário 1.

Container	ID 0	ID 1	ID 2	ID 3	ID 4
	0	1	2	3	4
	5	6	7	8	9
Cloudlet ID	10	11	12	13	14
	15	16	17	18	19
	20	21	22	23	24

Tabela 6: Arranjo de cloudlets em seus contêineres.

segundos, o consumo de memória é baixo ao decorrer de toda a simulação. É importante destacar que estão todos rodando na mesma máquina, mas com cada um utilizando sua reserva de recursos, o sistema consegue administrar bem os recursos e dividir os recursos compartilhados de forma justa entre os componentes.

5.2.2 Cenário 2

No segundo cenário pretendemos verificar o comportamento da arquitetura com menos carga. Portanto, desta vez colocamos 10 tarefas com apenas uma vCPU, e implantamos novamente em um data center, com 5 *hosts* e 3 containers, os outros valores podem ser vistos na Tabela 7.

Neste segundo cenário, com uma carga menor e menos competição de recursos, tivemos que alguns containers finalizaram o processamento de suas tarefas em um tempo menor que outros, por exemplo nas Figuras 18 e 19, eles terminaram antes do que na Figura 17.

Description	Values
Datacenter (Pod)	01
Hosts	05 with 04 vCPU
Container	03 with 02 vCPU reserved and 2048MB of RAM
Cloudlets (Queue)	10 with restriction to just 1 vCPU
Cloudlets (Instructions)	5.000 MIPS
Network Interface	10.000 Mbps

Tabela 7: Configuração da simulação - Cenário 2.

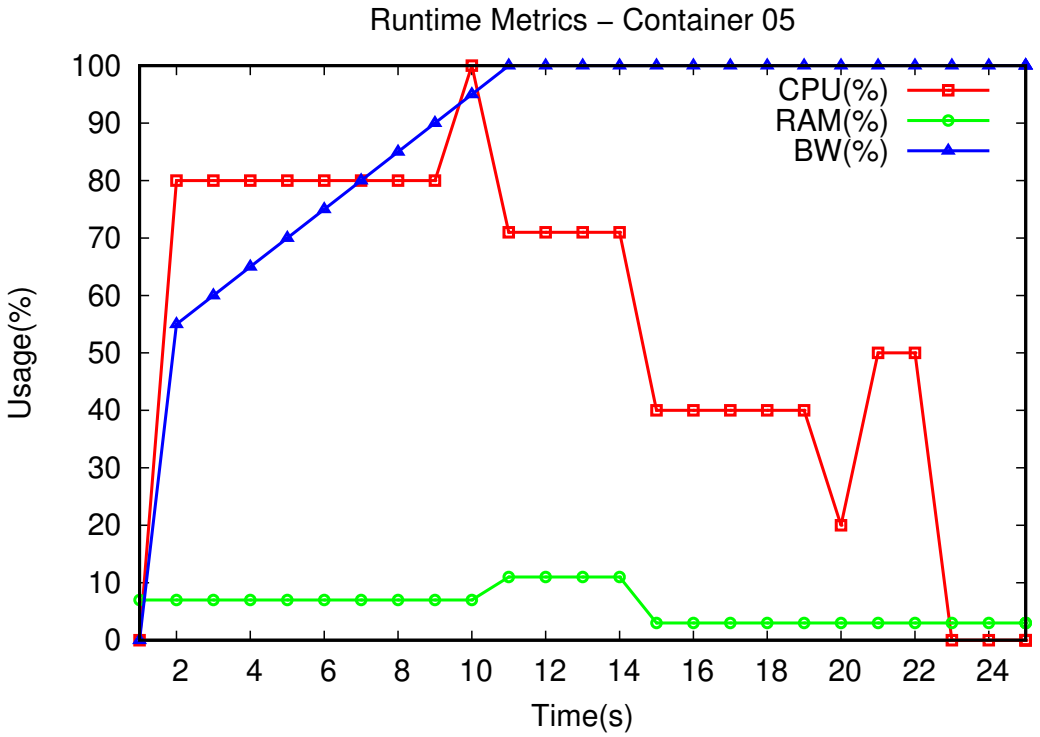


Figura 16: Implantação e execução do Container, no cenário 5.

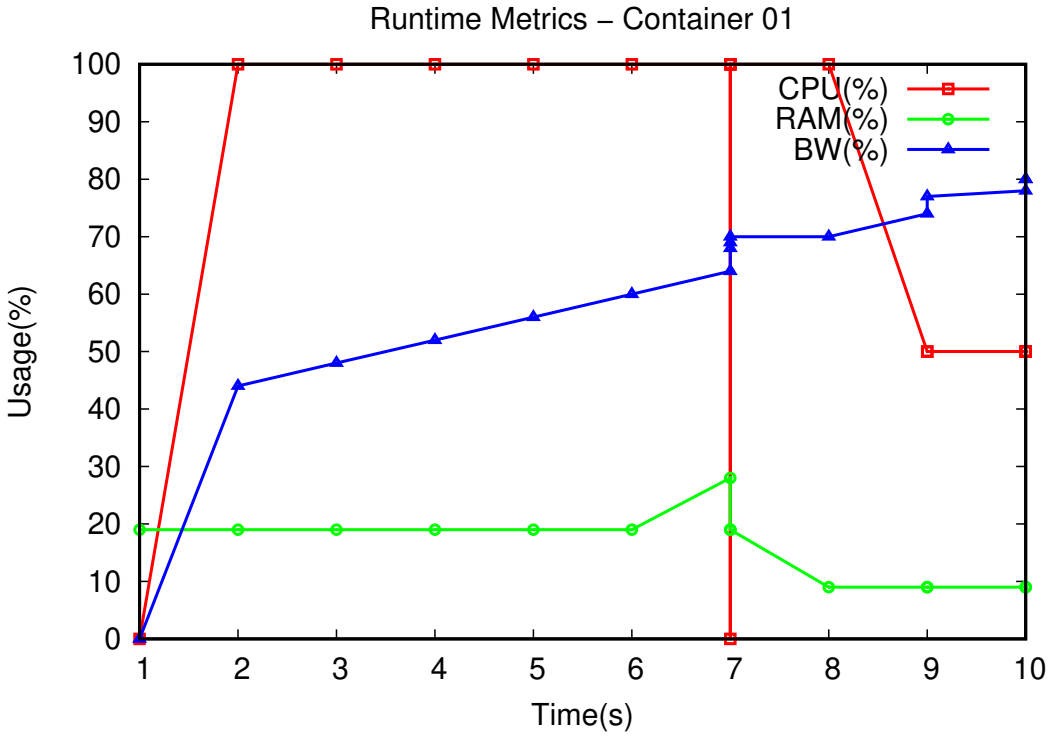


Figura 17: Implantação e execução do Container 1, no cenário 2

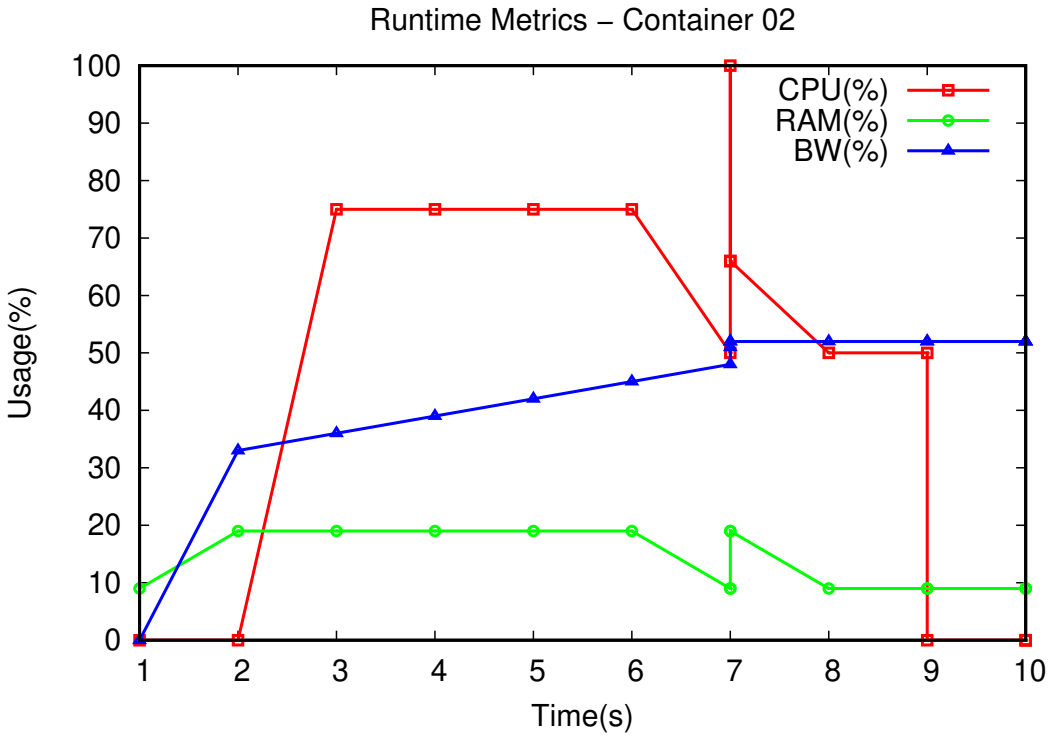


Figura 18: Implantação e execução do Container 2, no cenário 2.

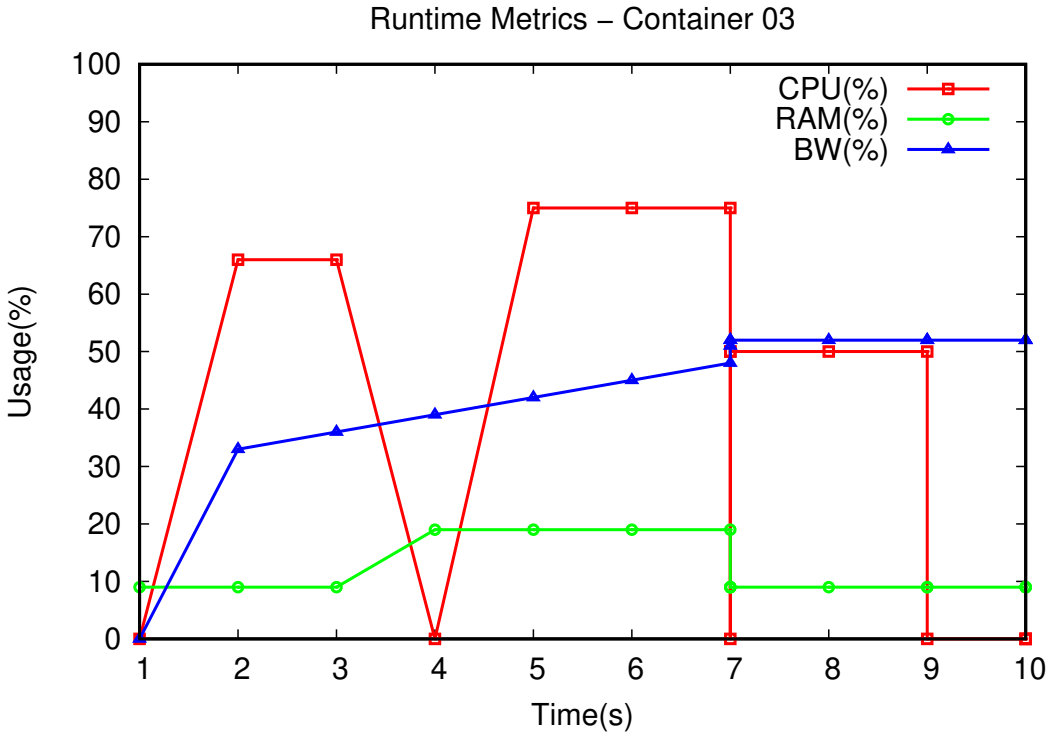


Figura 19: Implantação e execução do Container 3, no cenário 2.

Nestes dois cenários podemos verificar que o modelo proposto é capaz de atender de forma satisfatória ambientes com alta carga, compartilhando bem os recursos entre seus containers. Lembrando que não utilizamos nenhuma métrica de SLA ou QoS, todas as simulações foram atendidas com *Best Effort*.

5.3 Prova de Conceito

O cenário a seguir é proposto para a execução da prova de conceito. Em uma casa inteligente aprimorada com objetos de IoT capazes de detectar e agir em alguns ambientes, um sistema embarcado fornece coleta os dados dos sensores e, em seguida, envia as informações a um servidor de IoT. Além disso, recebe comandos de aplicativos para serem executados por Objetos IoT.

Usamos um protótipo de casa inteligente que é composto de sensores de temperatura, luminosidade e ultrassônicos, como podemos ver na Figura 20. Além disso, controla as luzes e um ventilador de teto e abre e fecha a janela.

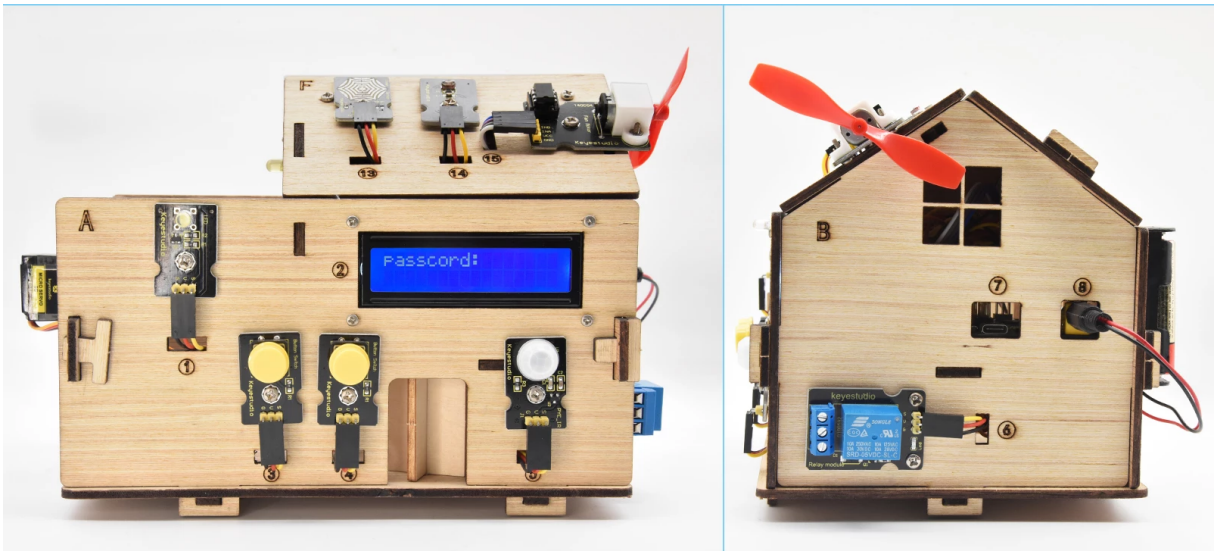


Figura 20: Teste com um protótipo de casa inteligente com sensores e atuadores.

Como os objetos IoT enviam dados para um servidor IoT, adotamos a Sistema de Gerenciamento de Recursos (SGR), que é baseado no RMA (75), uma arquitetura de três camadas para virtualização de objetos IoT a serem consumidos por clientes e aplicativos de terceiros. A conexão dos dados dos sensores e o serviço de gerenciamento de recursos foi realizada utilizando o middleware ContextNet. Além disso, o sistema expõe os dados dos Objetos IoT para clientes e outros aplicativos na camada Cliente.

Um dos desafios do desenvolvimento de sistemas para ambientes de IoT é que não é

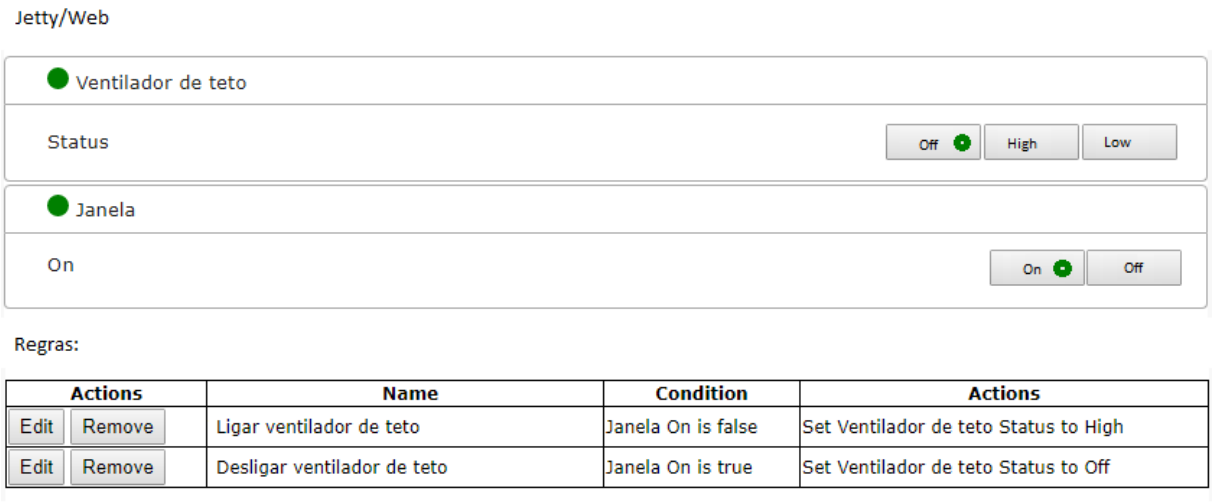


Figura 21: Tela da aplicação rodando no Jetty

simples implantar esse tipo de aplicativo, atuação em ambientes reais apresenta problemas de sincronização e comunicação. Além disso, é necessário que um servidor IoT na web ou *cloud*. O sistema deve estar disponível na Internet para testar a comunicação e sincronização de objetos IoT distantes e acesso de clientes. Por exemplo, quando o proprietário sai da casa inteligente, às vezes, ele pode esquecer algo ligado. Assim, ele pode verificar em um aplicativo se tudo está conforme o esperado. Em outro caso, um objeto IoT da sala de estar pode precisar trocar informações com um objeto IoT do quarto.

Não é trivial fornecer um ambiente de desenvolvimento para implementação e teste de aplicativos IoT. Para testar o sistema vamos implantar um sistema simples para controle do ventilador da Casa Inteligente, onde usaremos condições lógicas, onde a janela aberta desliga o ventilador e a janela fechada liga. As imagens abaixo apresentam a interface implementada no Jetty, que é um webservice RESTFull.

A imagem do site na Figura 21 apresenta a tela da aplicação web e sua execução, é possível visualizar os dados disponíveis, no cado os objetos Ventilador de Teto e Janela. Sobre os dispositivos, eles são cadastrados no SGR onde temos também a informação de meta dados acerta do *status* de valores dos atributos dos sensores, exemplo "se a janela está aberta". Logo abaixo na mesma imagem temos as regras a serem aplicadas nos objetos do ambiente, mesmo sendo um exemplo de aplicação reativa, é possível extrapolar para a implementação de algum sistema de inferência ou inteligência computacional.

A Figura 22 mostra os componentes referentes as camadas Monitor, NFVI, TestBed Service. O cAdivisor (19) é um *deamon* que roda nas imagens dos contêineres em execução e fornece aos usuários a visão dos recursos, representado na arquitetura a camada Monitor,

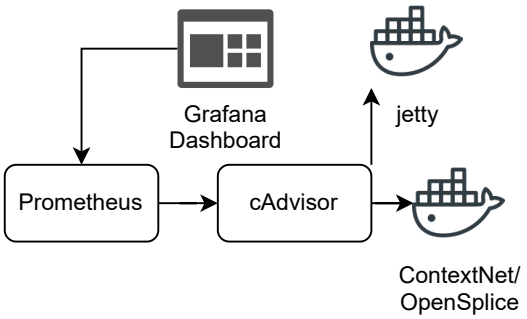


Figura 22: Diagrama dos Componentes para Rodar a Aplicação.

as informações fornecidas são sobre os processos em execução, histórico e estatísticas do uso de memória, disco, processamento e rede. Junto com o Prometheus(96) são os responsáveis pelo monitoramento da execução da aplicação. No Apêndice A consta as imagens dos deploys de containers, e o monitoramento durante a execução dos testes.

6 Conclusão

Neste trabalho, desenvolvemos uma arquitetura para implantação de sistemas ubíquos em nuvem, com o foco na oferta de um ambiente de testes como serviços, e na criação de ambientes virtuais para realização de experimentos. Com a possibilidade de compartilhamento dos dados de outros experimentos. A proposta pode ser utilizado na criação de novos sistemas. Usando as potencialidades que o modelo *Sensor-as-a-Service* fornece, e da arquitetura proposta, centros de pesquisa podem se favorecer do compartilhamento de dados e de infraestrutura para criação de nuvens para execução de experimentos. Pois a colaboração

Utilizando-se a arquitetura proposta, é possível criar um ambiente inteligente escalável apoiado por um *middleware* que permite a comunicação entre diversos dispositivos móveis em tempo real. Além disso, os sistemas construídos podem agregar dados de sensores de diversas fontes, sendo elas sensores reais, gerados de forma sintética, ou utilizando esquemas de *traces* públicos. Com o recurso de virtualização por contêineres, é possível replicar o experimento de forma mais amigável, bem como o procedimento de migração dos testes para outros servidores/nuvens.

Com a simulação da arquitetura foi possível verificar a capacidade de escalabilidade da proposta, problema que sempre tem uma função chave na avaliação de desempenho de sistemas distribuídos. Além da implementação das provas de conceito que possibilitaram a validação prática que são os motivadores da realização deste trabalho.

Ainda analisamos através de uma modelagem e simulação que é possível uma escalabilidade e atendimento de uma grande quantidade de requisições e deploys de aplicações, outro recurso importante é o registro da execução dos testes e o armazenamento das imagens e operações realizadas durante uma execução. A automatização da construção da infraestrutura de serviços também é uma importante contribuição que foi agregada a nossa arquitetura, diversas tecnologias de DevOps puderam ser integradas e fazem com que possamos ter um ganho quanto a reprodutibilidade e a escala das aplicações.

Na prova de conceito foi realizada a implementação de um serviço para atender algumas necessidades de um protótipo de casa inteligente, onde os sensores estão disponibilizados para acesso através de requisições a webservice. Outra vantagem é que para todas as aplicações que são executadas sobre a plataforma de testes, tantos os dados gerados, quanto todas as informações de imagens das execuções são salvas e podem ser reutilizadas por outros. Sendo assim, é possível concluir que as propostas apresentadas neste trabalho são consistentes e promissoras, visto que além dos bons resultados alcançados, o modelo matemático possui uma estrutura simples e que basicamente necessita de poucos parâmetros de calibração.

6.1 Limitações

No estado atual do trabalho algumas limitações foram identificadas. Usando nuvens para disponibilizar aplicações se faz necessário a avaliação da proposta em nuvens comerciais como AWS, Azure entre outras, isso possibilita uma solução de engenharia alinhada com modelos mais comerciais. Já a parte de modelagem é notório salientar a necessidade de modelar o comportamento do ponto de vista de consumo energético e também o funcionamento desta proposta em ambientes *multi-cloud*, a partir dos modelos já desenvolvidos podemos expandir para incluir esses novos requisitos.

O sistema de gerência de recursos SGR, foi utilizado para os testes implementando algumas de suas funções por codificação rígida, é importante apontar que o ambiente *Front-end* necessita de reformulação para atender requisitos funcionais e não funcionais de interface, bem como um controle de usuário por níveis e possível extensão para controle modelo de serviço de testbed federado como em (13).

Outro problema é que mesmo usando o SGR para gerenciar dispositivos e recursos, quando se trata de integração de dados de diferentes ambiantes ainda existe a questão de harmonização de dados, pois o mesmo dado pode ser cadastrado e formatado de forma diferente por diferentes programadores ou fontes de dados. Contudo é possível estender a arquitetura para que tenha um sistema de *Data Models* (63), tanto para o desenvolvimento próprio como a adoção de um sistema já existente.

6.2 Trabalhos Futuros

Trabalhos futuros incluem a simulação de casos reais, de modo a validar os modelos propostos. De forma adicional, pode-se otimizar recursos computacionais além dos aqui tratados, tais como consumo de energia e balanceamento das cargas de trabalho. Outra possibilidade incluí a utilização de heurísticas para tratar os problemas de alocação de recursos em tempo real, visto que o elevado custo computacional exigido por técnicas exatas torna inviável, em muitos casos, a resolução deste tipo de problema.

REFERÊNCIAS

1 ABERER, Karl; HAUSWIRTH, Manfred; SALEHI, Ali. A middleware for fast and flexible sensor network deployment. In: VLDB ENDOWMENT. PROCEEDINGS of the 32nd international conference on Very large data bases. [S. l.: s. n.], 2006. p. 1199–1202.

2 ADJIH, Cedric et al. FIT IoT-LAB: A large scale open experimental IoT testbed. **IEEE World Forum on Internet of Things, WF-IoT 2015 - Proceedings**, p. 459–464, 2015. DOI: 10.1109/WF-IoT.2015.7389098.

3 AGRAWAL, Divyakant; DAS, Sudipto; EL ABBADI, Amr. Big data and cloud computing: new wine or just new bottles? **Proceedings of the VLDB Endowment**, VLDB Endowment, v. 3, n. 1-2, p. 1647–1648, 2010.

4 AMARAL, Leonardo Albernaz et al. Cooperative Middleware Platform as a Service for Internet of Things Applications. In: PROCEEDINGS of the 30th Annual ACM Symposium on Applied Computing. Salamanca, Spain: Association for Computing Machinery, 2015. (SAC '15), p. 488–493. ISBN 9781450331968. DOI: 10.1145/2695664.2695799. Disponível em: <https://doi.org/10.1145/2695664.2695799>.

5 ARMAC, Ibrahim; RETKOWITZ, Daniel. Simulation of smart environments. In: IEEE. IEEE International Conference on Pervasive Services. [S. l.: s. n.], 2007. p. 257–266. DOI: 10.1109/PERSER.2007.4283934.

6 ARREGOCES, Mauricio; PORTOLANI, Maurizio. **Data center fundamentals**. [S. l.]: Cisco Press, 2003.

7 ASAEDA, Hitoshi; LI, Ruidong; CHOI, Nakjung. Container-based unified testbed for information-centric networking. **IEEE Network**, IEEE, v. 28, n. 6, p. 60–66, 2014. ISSN 08908044. DOI: 10.1109/MNET.2014.6963806.

8 ASHTON, Kevin et al. That 'internet of things' thing. **RFID journal**, Jun, v. 22, n. 7, p. 97–114, 2009.

9 BANERJEE, Prith et al. Everything as a service: Powering the new information economy. **Computer**, IEEE, n. 3, p. 36–43, 2011.

10 BARKER, Sean et al. Smart*: An open data set and tools for enabling research in sustainable homes. **SustKDD, August**, v. 111, n. 112, p. 108, 2012.

11 BARSHAN, Maryam et al. Algorithms for network-aware application component placement for cloud resource allocation. **Journal of Communications and Networks**, v. 19, n. 5, p. 493–508, 2017. DOI: 10.1109/JCN.2017.000081.

12 BASS, Len; CLEMENTS, Paul; KAZMAN, Rick. **Software architecture in practice**. [S. l.]: Addison-Wesley Professional, 2003.

13 BERMAN, Mark et al. GENI: A federated testbed for innovative network experiments. **Computer Networks**, Elsevier, v. 61, p. 5–23, 2014.

14 BERNSTEIN, David. Containers and cloud: From lxc to docker to kubernetes. **IEEE Cloud Computing**, IEEE, v. 1, n. 3, p. 81–84, 2014. DOI: 10.1109/MCC.2014.51.

15 BESERRA, David et al. Performance analysis of LXC for HPC environments. In: IEEE. 2015 Ninth International Conference on Complex, Intelligent, and Software Intensive Systems. [S. l.: s. n.], 2015. p. 358–363.

16 BHARDWAJ, Sushil; JAIN, Leena; JAIN, Sandeep. Cloud computing: A study of infrastructure as a service (IAAS). **International Journal of engineering and information Technology**, v. 2, n. 1, p. 60–63, 2010.

17 BOETTIGER, Carl. An introduction to Docker for reproducible research. **ACM SIGOPS Operating Systems Review**, ACM, v. 49, n. 1, p. 71–79, 2015.

18 CARAGLIU, Andrea; DEL BO, Chiara; NIJKAMP, Peter. Smart cities in Europe. **Journal of urban technology**, Taylor & Francis, v. 18, n. 2, p. 65–82, 2011.

19 CASALICCHIO, Emiliano; PERCIBALLI, Vanessa. Measuring docker performance: What a mess!!! In: PROCEEDINGS of the 8th ACM/SPEC on International Conference on Performance Engineering Companion. [S. l.: s. n.], 2017. p. 11–16.

20 CELESTI, Antonio et al. A watchdog service making container-based micro-services reliable in IoT clouds. **Proceedings - 2017 IEEE 5th International Conference on Future Internet of Things and Cloud, FiCloud 2017**, 2017-Janua, n. 55, p. 372–378, 2017. DOI: 10.1109/FiCloud.2017.57.

21 CELESTI, Antonio et al. Exploring Container Virtualization in IoT Clouds. **2016 IEEE International Conference on Smart Computing, SMARTCOMP 2016**, IEEE, n. 140, p. 1–6, 2016. DOI: 10.1109/SMARTCOMP.2016.7501691.

22 CHEN, Dong et al. Non-intrusive occupancy monitoring using smart meters. In: ACM. PROCEEDINGS of the 5th ACM Workshop on Embedded Systems For Energy-Efficient Buildings. [S. l.: s. n.], 2013. p. 1–8.

23 CHEN, H. et al. Intelligent agents meet the semantic Web in smart spaces. **IEEE Internet Computing**, v. 8, n. 6, p. 69–79, 2004. DOI: 10.1109/MIC.2004.66.

24 CHEN, Harry. **An Intelligent Broker Architecture for Pervasive Context-Aware Systems**. Dez. 2004. Tese (Doutorado) – University of Maryland, Baltimore County.

25 CHUN, Brent et al. PlanetLab: An Overlay Testbed for Broad-coverage Services. **SIGCOMM Comput. Commun. Rev.**, ACM, New York, NY, USA, v. 33, n. 3, p. 3–12, jul. 2003. ISSN 0146-4833. DOI: 10.1145/956993.956995. Disponível em: <http://doi.acm.org/10.1145/956993.956995>.

26 _____. Planetlab: an overlay testbed for broad-coverage services. **ACM SIGCOMM Computer Communication Review**, ACM, v. 33, n. 3, p. 3–12, 2003.

27 CIRILLO, Flavio et al. A standard-based open source IoT platform: FIWARE. **IEEE Internet of Things Magazine**, IEEE, v. 2, n. 3, p. 12–18, 2019.

28 CONTEXTNET. **Laboratory for Advanced Collaboration**. [S. l.: s. n.], 2021. Disponível em: <www.lac.inf.puc-rio.br/index.php/softwares-contextnet/>. Acesso em: 11 nov. 2021.

29 COOK, D. J. et al. CASAS: A Smart Home in a Box. **Computer**, v. 46, n. 7, p. 62–69, jul. 2013. ISSN 0018-9162. DOI: 10.1109/MC.2012.328.

30 COOK, Diane et al. Collecting and disseminating smart home sensor data in the CASAS project. In: PROCEEDINGS of the CHI workshop on developing shared home behavior datasets to advance HCI and ubiquitous computing research. [S. l.: s. n.], 2009. p. 1–7.

31 COULOURIS, George F; DOLLIMORE, Jean; KINDBERG, Tim. **Distributed systems: concepts and design**. [S. l.]: pearson education, 2005.

32 DA XU, Li; HE, Wu; LI, Shancang. Internet of things in industries: A survey. **IEEE Transactions on industrial informatics**, IEEE, v. 10, n. 4, p. 2233–2243, 2014. DOI: 10.1109/TII.2014.2300753.

33 DAS, Sajal K et al. The role of prediction algorithms in the MavHome smart home architecture. **IEEE Wireless Communications**, IEEE, v. 9, n. 6, p. 77–84, 2002. DOI: 10.1109/MWC.2002.1160085.

34 DAVID, Lincoln et al. A DDS-based middleware for scalable tracking, communication and collaboration of mobile nodes. **Journal of Internet Services and Applications**, v. 4, n. 1, p. 16, 2013. DOI: https://doi.org/10.1007/978-3-540-78731-0_4.

35 DE SOUZA, Luciana Moreira Sá et al. Socrades: A web service based shop floor integration infrastructure. In: THE internet of things. [S. l.]: Springer, 2008. p. 50–67.

36 DOCKER, INCORPORATED. **Docker**. [S. l.: s. n.]. https://www.docker.com. Acessado em 19-03-2019.

37 DUA, Rajdeep; RAJA, A Reddy; KAKADIA, Dharmesh. Virtualization vs containerization to support paas. In: IEEE. 2014 IEEE International Conference on Cloud Engineering. [S. l.: s. n.], 2014. p. 610–614.

38 DUAN, Yucong et al. Everything as a service (XaaS) on the cloud: origins, current and future trends. In: IEEE. 2015 IEEE 8th International Conference on Cloud Computing. [S. l.: s. n.], 2015. p. 621–628.

39 ELNOZAHY, E. N.; KISTLER, Michael; RAJAMONY, Ramakrishnan. Energy-Efficient Server Clusters. In: [s. l.: s. n.], 2003. p. 179–197. DOI: 10.1007/3-540-36612-1_12. Disponível em: <http://link.springer.com/10.1007/3-540-36612-1_12>.

40 ENDLER, M. et al. ContextNet: Context Reasoning and Sharing Middleware for Large-scale Pervasive Collaboration and Social Networking. In: PROCEEDINGS of the Workshop on Posters and Demos Track. Lisbon, Portugal: ACM, 2011. (PDT '11), 2:1–2:2. ISBN 978-1-4503-1073-4. DOI: 10.1145/2088960.2088962. Disponível em: <http://doi.acm.org/10.1145/2088960.2088962>.

41 ENDLER, Markus; SILVA, Francisco Silva e. Past, present and future of the contextnet iomt middleware. **Open Journal of Internet Of Things (OJIOT)**, RonPub, v. 4, n. 1, p. 7–23, 2018.

42 EUGSTER, Patrick. Type-based publish/subscribe: Concepts and experiences. **ACM Transactions on Programming Languages and Systems (TOPLAS)**, ACM New York, NY, USA, v. 29, n. 1, 6–es, 2007.

43 FORKAN, Abdur; KHALIL, Ibrahim; TARI, Zahir. CoCaMAAL: A cloud-oriented context-aware middleware in ambient assisted living. **Future Generation Computer Systems**, v. 35, p. 114–127, 2014. Special Section: Integration of Cloud

Computing and Body Sensor Networks; Guest Editors: Giancarlo Fortino and Mukaddim Pathan. ISSN 0167-739X. DOI: https://doi.org/10.1016/j.future.2013.07.009. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0167739X13001544>.

44 GLUHAK, A. et al. A survey on facilities for experimental internet of things research. **IEEE Communications Magazine**, v. 49, n. 11, p. 58–67, nov. 2011. ISSN 0163-6804. DOI: 10.1109/MCOM.2011.6069710.

45 GU, Tao; PUNG, Hung Keng; ZHANG, Da Qing. A service-oriented middleware for building context-aware services. **Journal of Network and Computer Applications**, v. 28, n. 1, p. 1–18, 2005. ISSN 1084-8045. DOI: https://doi.org/10.1016/j.jnca.2004.06.002. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1084804504000451>.

46 GUINARD, Dominique; TRIFA, Vlad. Towards the web of things: Web mashups for embedded devices. In: WORKSHOP on Mashups, Enterprise Mashups and Lightweight Composition on the Web (MEM 2009), in proceedings of WWW (International World Wide Web Conferences), Madrid, Spain. [S. l.: s. n.], 2009. v. 15.

47 HELAL, A.; MENDEZ-VAZQUEZ, A.; HOSSAIN, S. Specification and synthesis of sensory datasets in pervasive spaces. In: 2009 IEEE Symposium on Computers and Communications. [S. l.: s. n.], jul. 2009. p. 920–925. DOI: 10.1109/ISCC.2009.5202263.

48 HIGHTOWER, Kelsey; BURNS, Brendan; BEDA, Joe. **Kubernetes: Up and Running: Dive Into the Future of Infrastructure**. [S. l.]: "O'Reilly Media, Inc.", 2017.

49 HUANG, Yu Lun et al. Security impacts of virtualization on a network testbed. **Proceedings of the 2012 IEEE 6th International Conference on Software Security and Reliability, SERE 2012**, IEEE, p. 71–77, 2012. DOI: 10.1109/SERE.2012.17.

50 IBBOTSON, John et al. Sensors as a service oriented architecture: Middleware for sensor networks. In: IEEE. 2010 Sixth International Conference on Intelligent Environments. [S. l.: s. n.], 2010. p. 209–214.

51 JACOBSON, Van et al. Networking named content. In: ACM. PROCEEDINGS of the 5th international conference on Emerging networking experiments and technologies. [S. l.: s. n.], 2009. p. 1–12.

52 KAUR, Kuljeet et al. Container-as-a-service at the edge: Trade-off between energy efficiency and service availability at fog nano data centers. **IEEE wireless communications**, IEEE, v. 24, n. 3, p. 48–56, 2017.

53 KDOUH, H. et al. Performance analysis of a hierarchical shipboard Wireless Sensor Network. In: 2012 IEEE 23rd International Symposium on Personal, Indoor and Mobile Radio Communications - (PIMRC). [S. l.: s. n.], set. 2012. p. 765–770. DOI: 10.1109/PIMRC.2012.6362886.

54 KIDD, Cory D et al. The aware home: A living laboratory for ubiquitous computing research. In: SPRINGER. INTERNATIONAL Workshop on Cooperative Buildings. [S. l.: s. n.], 1999. p. 191–198.

55 KILJANDER, Jussi et al. Semantic interoperability architecture for pervasive computing and internet of things. **IEEE access**, IEEE, v. 2, p. 856–873, 2014.

56 LAZARIN, Nilson Mori; PANTOJA, Carlos Eduardo. A robotic-agent platform for embedding software agents using raspberry pi and arduino boards. **9th Software Agents, Environments and Applications School**, 2015.

57 LECH, Till C.; WIENHOFEN, Leendert W. M. AmbieAgents: A Scalable Infrastructure for Mobile and Context-Aware Information Services. In: PROCEEDINGS of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems. The Netherlands: Association for Computing Machinery, 2005. (AAMAS '05), p. 625–631. ISBN 1595930930. DOI: 10.1145/1082473.1082568. Disponível em: <https://doi.org/10.1145/1082473.1082568>.

58 LESSER, Victor et al. The intelligent home testbed. **environment**, Citeseer, v. 2, p. 15, 1999.

59 LI, Shancang; DA XU, Li; ZHAO, Shanshan. The internet of things: a survey. **Information Systems Frontiers**, Springer, v. 17, n. 2, p. 243–259, 2015.

60 LIEN, Chia-Hung; BAI, Ying-Wen; LIN, Ming-Bo. Estimation by Software for the Power Consumption of Streaming-Media Servers. **IEEE Transactions on Instrumentation and Measurement**, v. 56, n. 5, p. 1859–1870, out. 2007. ISSN 0018-9456. DOI: 10.1109/TIM.2007.904554. Disponível em: <http://ieeexplore.ieee.org/document/4303423/>.

61 LOPEZ, Jorge. **Building and Expanding a Digital Business Primer for 2019**. [S. l.: s. n.], 2019. https://www.gartner.com/doc/3898265. Acessado em 28-03-2019.

62 MATOS, Everton de et al. A sensing-as-a-service context-aware system for Internet of Things environments. In: 2017 14th IEEE Annual Consumer Communications Networking Conference (CCNC). [S. l.: s. n.], 2017. p. 724–727. DOI: 10.1109/CCNC.2017.7983223.

63 MILENKOVIC, Milan. IoT Data Models and Metadata. In: INTERNET of Things: Concepts and System Design. [S. l.]: Springer, 2020. p. 201–223.

64 MISHRA, Aditya et al. Smartcharge: Cutting the electricity bill in smart homes with energy storage. In: ACM. PROCEEDINGS of the 3rd International Conference on Future Energy Systems: Where Energy, Computing and Communication Meet. [S. l.: s. n.], 2012. p. 29.

65 MORABITO, Roberto. A performance evaluation of container technologies on Internet of Things devices. **Proceedings - IEEE INFOCOM**, IEEE, 2016-Septe, p. 999–1000, 2016. ISSN 0743166X. DOI: 10.1109/INFCOMW.2016.7562228.

66 _____. Virtualization on internet of things edge devices with container technologies: A performance evaluation. **IEEE Access**, IEEE, v. 5, p. 8835–8850, 2017. ISSN 21693536. DOI: 10.1109/ACCESS.2017.2704444. eprint: 1603.02955.

67 MORABITO, Roberto et al. Consolidate IoT Edge Computing with Lightweight Virtualization. **IEEE Network**, v. 32, February, p. 102–111, 2018. ISSN 08908044. DOI: 10.1109/MNET.2018.1700175. Disponível em: <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=%7B%5C&%7Darnumber=8270640%7B%5C%%7D0Ahttps://ieeexplore.ieee.org/document/8270640/>.

68 MOZER, Michael C. Lessons from an Adaptive Home. In: SMART Environments. [S. l.]: John Wiley & Sons, Ltd, 2005. cap. 12, p. 271–294. ISBN 9780471686590. DOI: 10.1002/047168659X.ch12. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/047168659X.ch12. Disponível em: <https://onlinelibrary.wiley.com/doi/abs/10.1002/047168659X.ch12>.

69 NEIROTTI, Paolo et al. Current trends in Smart City initiatives: Some stylised facts. **Cities**, Elsevier, v. 38, p. 25–36, 2014.

70 NGO, Hung Q. et al. Developing Context-Aware Ubiquitous Computing Systems with a Unified Middleware Framework. In_____. **Embedded and Ubiquitous Computing**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004. p. 672–681. DOI: 10.1007/978-3-540-30121-9_64.

71 NORBISRATH, Ulrich; SALUMAA, Priit; MALIK, Adam. **eHome specification, configuration, and deployment**. [S. l.: s. n.], 2005.

72 OSTERMAIER, B.; SCHLUP, F.; RÖMER, K. WebPlug: A framework for the Web of Things. In: 2010 8th IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops). [S. l.: s. n.], mar. 2010. p. 690–695. DOI: 10.1109/PERCOMW.2010.5470522.

73 PANTOJA, Carlos; SOUZA DE JESUS, Vinicius; VITERBO, Jose. Aplicando Sistemas Multi-Agentes Ubíquos em um Modelo de Smart Home Usando o Framework Jason. In: II Workshop de Pesquisa e Desenvolvimento em Inteligência Artificial, Inteligência Coletiva e Ciência de Dados. [S. l.: s. n.], dez. 2016.

74 PANTOJA, Carlos Eduardo; SOARES, Heder Dorneles; VITERBO, José. A Resource Management Architecture For Exposing Devices as a Service in the Internet of Things. unpublished. [S. l.], 2019.

75 PANTOJA, Carlos Eduardo et al. An Architecture for the Development of Ambient Intelligence Systems Managed by Embedded Agents. In: THE 30th International Conference on Software Engineering & Knowledge Engineering. [S. l.: s. n.], 2018. p. 215–220.

76 PANWAR, Nisha; SHARMA, Shantanu; SINGH, Awadhesh Kumar. A survey on 5G: The next generation of mobile communication. **Physical Communication**, v. 18, p. 64–84, 2016. ISSN 1874-4907. DOI: https://doi.org/10.1016/j.phycom.2015.10.006. Disponível em: <http://www.sciencedirect.com/science/article/pii/S1874490715000531>.

77 PAPADOPOULOS, Georgios Z et al. Thorough IoT testbed characterization: From proof-of-concept to repeatable experimentations. **Computer Networks**, Elsevier, v. 119, p. 86–101, 2017.

78 PARDO-CASTELLOTE, Gerardo. Omg data-distribution service: Architectural overview. In: IEEE. DISTRIBUTED Computing Systems Workshops, 2003. Proceedings. 23rd International Conference on. [S. l.: s. n.], 2003. p. 200–206.

79 PAULSON, L. D. Building rich web applications with Ajax. **Computer**, v. 38, n. 10, p. 14–17, out. 2005. ISSN 0018-9162. DOI: 10.1109/MC.2005.330.

80 PERERA, Charith et al. Sensing as a Service Model for Smart Cities Supported by Internet of Things. **Transactions on Emerging Telecommunications Technologies**, v. 25, n. 3, p. 294–307, 2014. ISSN 15418251. DOI: 10.1002/ett. eprint: arXiv:1307.8198v1.

81 PIRAGHAJ, Sareh Fotuhi et al. Efficient virtual machine sizing for hosting containers as a service (services 2015). In: IEEE. 2015 IEEE World Congress on Services. [S. l.: s. n.], 2015. p. 31–38.

82 QIN, Weijun; SUO, Yue; SHI, Yuanchun. CAMPS: A Middleware for Providing Context-Aware Services for Smart Space. In: PROCEEDINGS of the First International Conference on Advances in Grid and Pervasive Computing. Taichung, Taiwan: [s. n.], 2006. (GPC'06), p. 644–653. ISBN 3540338098. DOI: 10.1007/11745693_63. Disponível em: <https://doi.org/10.1007/11745693_63>.

83 ROMAN, M. et al. A middleware infrastructure for active spaces. **IEEE Pervasive Computing**, v. 1, n. 4, p. 74–83, 2002. DOI: 10.1109/MPRV.2002.1158281.

84 ROSEN, Rami. Linux containers and the future cloud. **Linux J**, v. 240, n. 4, p. 86–95, 2014.

85 SANTOS, E. et al. Logistics SLA Optimization Service for Transportation in Smart Cities. In: 2018 International Joint Conference on Neural Networks (IJCNN). [S. l.: s. n.], jul. 2018. p. 1–8. DOI: 10.1109/IJCNN.2018.8489344.

86 SCHANTZ, Richard E; SCHMIDT, Douglas C. Middleware for distributed systems: Evolving the common structure for network-centric applications. **Encyclopedia of Software Engineering**, Citeseer, v. 1, p. 1–9, 2001.

87 SCHILIT, Bill N.; ADAMS, Norman; WANT, Roy. Context-Aware Computing Applications. In: IN PROCEEDINGS OF THE WORKSHOP ON MOBILE COMPUTING SYSTEMS AND APPLICATIONS. [S. l.]: IEEE Computer Society, 1994. p. 85–90.

88 SCHMIDT, Rainer et al. Industry 4.0 - Potentials for Creating Smart Products: Empirical Research Results. In_____. **Business Information Systems**. Cham: Springer International Publishing, 2015. p. 16–27. ISBN 978-3-319-19027-3.

89 SHAW, Mary. Toward higher-level abstractions for software systems. **Data & Knowledge Engineering**, Elsevier, v. 5, n. 2, p. 119–128, 1990.

90 SIFALAKIS, Manolis et al. An information centric network for computing the distribution of computations. In: ACM. PROCEEDINGS of the 1st ACM Conference on Information-Centric Networking. [S. l.: s. n.], 2014. p. 137–146.

91 SILVA, Edgar M; MALÓ, Pedro. IoT testbed business model. **Advances in Internet of Things**, Scientific Research Publishing, v. 4, n. 04, p. 37, 2014.

92 SILVA, LD; ENDLER, M; RORIZ, M. MR-UDP: Yet another reliable user datagram protocol, now for mobile nodes. **Monografias em Ciência da Computação, nr**, v. 1200, p. 06–13, 2013.

93 SILVA FILHO, Manoel C et al. CloudSim Plus: A cloud computing simulation framework pursuing software engineering principles for improved modularity, extensibility and correctness. In: IEEE. 2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM). [S. l.: s. n.], 2017. p. 400–406.

94 SINGLA, Geetika; COOK, Diane J; SCHMITTER-EDGECOMBE, Maureen. Recognizing independent and joint activities among multiple residents in smart environments. **Journal of ambient intelligence and humanized computing**, Springer, v. 1, n. 1, p. 57–63, 2010.

95 TERZIYAN, Vagan; KAYKOVA, Olena; ZHOVTOBRYUKH, Dmytro. UbiRoad: Semantic Middleware for Context-Aware Smart Road Environments. In: 2010 Fifth International Conference on Internet and Web Applications and Services. [S. l.: s. n.], 2010. p. 295–302. DOI: 10.1109/ICIW.2010.50.

96 TURNBULL, James. **Monitoring with Prometheus**. [S. l.]: Turnbull Press, 2018.

97 VASCONCELOS, Igor et al. Desenvolvendo Aplicações de Rastreamento e Comunicação Móvel usando o Middleware SDDL. In: SALÃO de Ferramentas, Brazilian Symposium on Computer Networks and Distributed Systems (SBRC 2013). [S. l.: s. n.], 2013.

98 VAUGHAN-NICHOLS, Stephen J. New approach to virtualization is a lightweight. **Computer**, IEEE, v. 39, n. 11, p. 12–14, 2006.

99 VÖGLER, Michael et al. A Scalable Framework for Provisioning Large-Scale IoT Deployments. **ACM Transactions on Internet Technology**, v. 16, n. 2, p. 1–20, 2016. ISSN 15335399. DOI: 10.1145/2850416.

100 WADLEY, Virginia G et al. Mild cognitive impairment and everyday function: evidence of reduced speed in performing instrumental activities of daily living. **The American Journal of Geriatric Psychiatry**, Elsevier, v. 16, n. 5, p. 416–424, 2008.

101 WANG, Qian et al. CS-Man: Computation service management for IoT in-network processing. **2016 27th Irish Signals and Systems Conference, ISSC 2016**, IEEE, p. 1–6, 2016. DOI: 10.1109/ISSC.2016.7528464.

102 WANT, Roy et al. The active badge location system. **ACM Transactions on Information Systems (TOIS)**, ACM, v. 10, n. 1, p. 91–102, 1992.

103 WEISER, Mark. Some Computer Science Issues in Ubiquitous Computing. **SIGMOBILE Mob. Comput. Commun. Rev.**, ACM, New York, NY, USA, v. 3, n. 3, p. 12–, jul. 1999. ISSN 1559-1662. DOI: 10.1145/329124.329127. Disponível em: <http://doi.acm.org/10.1145/329124.329127>.

104 _____. The Computer for the 21 st Century. **Scientific american**, JSTOR, v. 265, n. 3, p. 94–105, 1991.

105 XU, Jing; FORTES, Jose A. B. Multi-Objective Virtual Machine Placement in Virtualized Data Center Environments. In: 2010 IEEE/ACM International Conference on Green Computing and Communications AND International Conference on Cyber, Physical and Social Computing. [S. l.]: IEEE, dez. 2010. p. 179–188. ISBN 978-1-4244-9779-9. DOI: 10.1109/GreenCom-CPSCom.2010.137. Disponível em: <http://ieeexplore.ieee.org/document/5724828/>.

106 YOUNGBLOOD, G Michael; COOK, Diane J; HOLDER, Lawrence B. Managing adaptive versatile environments. **Pervasive and Mobile Computing**, Elsevier, v. 1, n. 4, p. 373–403, 2005.

107 YURIYAMA, Madoka; KUSHIDA, Takayuki. Sensor-cloud infrastructure-physical sensor management with virtualized sensors on cloud computing. In: IEEE. 2010 13th International Conference on Network-Based Information Systems. [S. l.: s. n.], 2010. p. 1–8.

108 ZHANG, Lixia et al. Named data networking. **ACM SIGCOMM Computer Communication Review**, ACM, v. 44, n. 3, p. 66–73, 2014.

109 ZHU, Ting et al. The case for efficient renewable energy management in smart homes. In: ACM. PROCEEDINGS of the Third ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings. [S. l.: s. n.], 2011. p. 67–72.

APÊNDICE A – EXECUÇÃO DO PROTÓTIPO

Neste apêndice constam as imagens da execução dos testes do protótipo de avaliação da arquitetura.

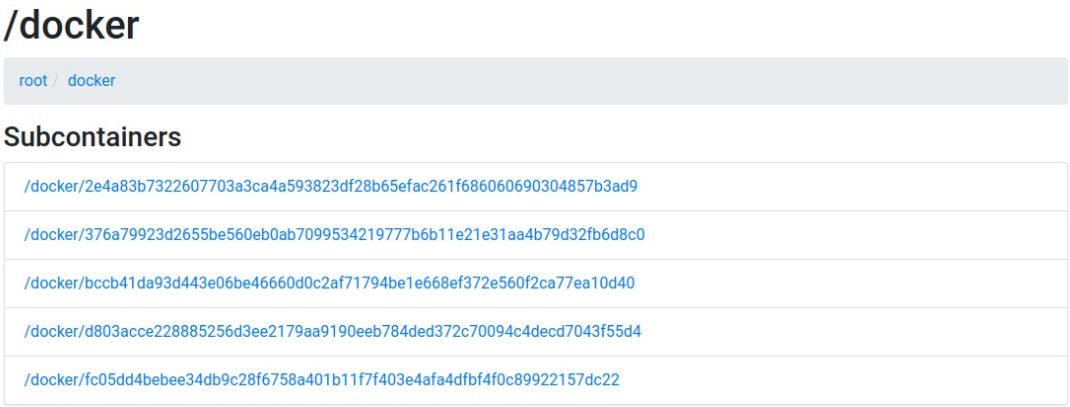


Figura 23: Imagens do docker em execução.

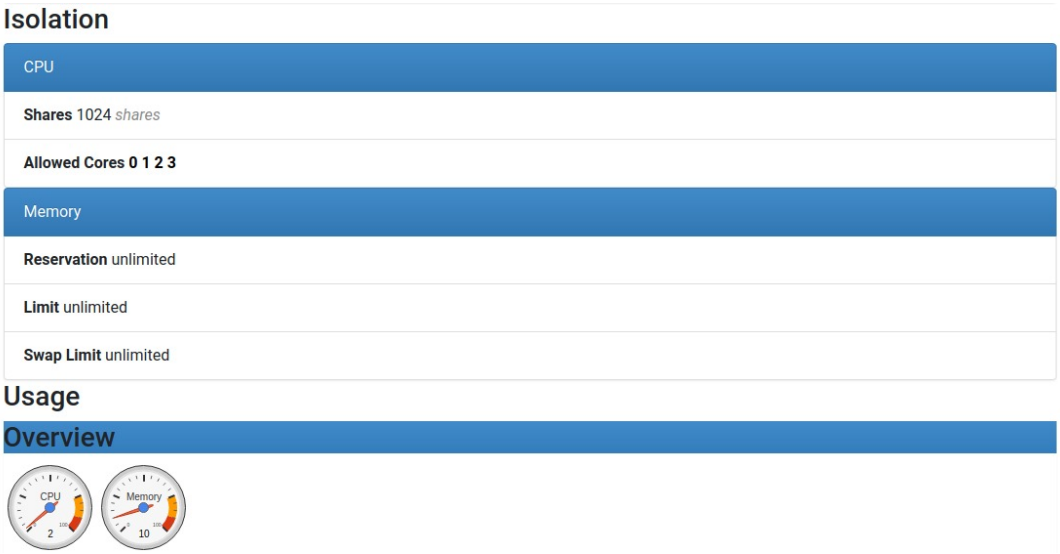


Figura 24: Consumo de uso.

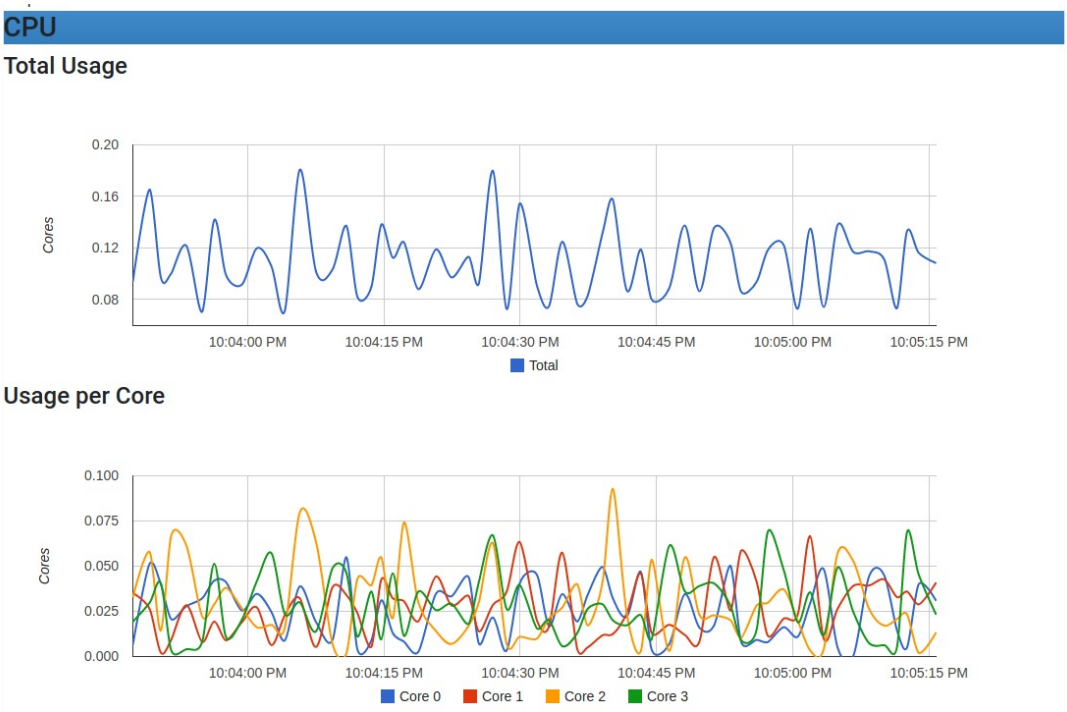


Figura 25: Uso de CPU.

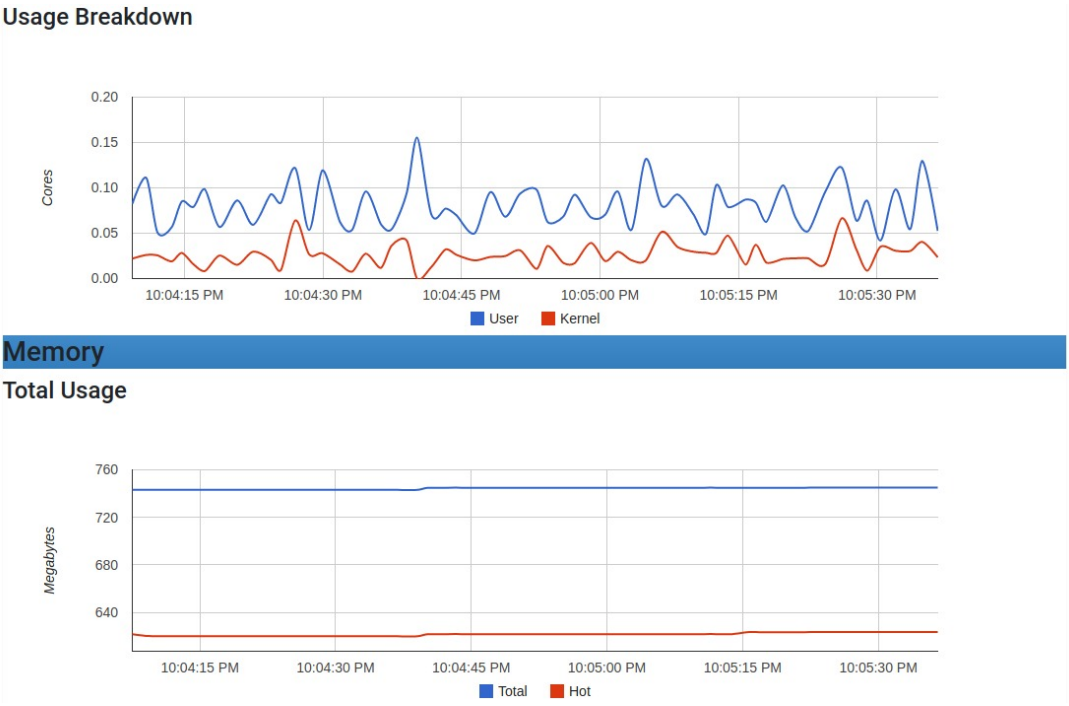


Figura 26: Uso de memória.