UNIVERSIDADE ESTADUAL DA PARAÍBA
CAMPUS VII – GOVERNADOR ANTÔNIO MARIZ
CENTRO DE CIÊNCIAS EXATAS E SOCIAIS APLICADAS
CURSO DE BACHARELADO EM COMPUTAÇÃO

ALLAN MEDEIROS DE LIMA

UMA ABORDAGEM PARA O GERENCIAMENTO DE INFRAESTRUTURAS VIRTUALIZADAS EM AMBIENTES DE INTERNET DAS COISAS

PATOS – PB
2019

ALLAN MEDEIROS DE LIMA

UMA ABORDAGEM PARA O GERENCIAMENTO DE INFRAESTRUTURAS VIRTUALIZADAS EM AMBIENTES DE INTERNET DAS COISAS

Trabalho de Conclusão de Curso apresentado ao Curso de Bacharelado em Computação da Universidade Estadual da Paraíba, em cumprimento à exigência para obtenção do grau de bacharel em computação.

Área de concentração: Ciência da Computação

Orientador: Prof. Dr. Rodrigo Alves Costa

PATOS – PB
2019

L732a Lima, Allan Medeiros de.

Uma abordagem para o gerenciamento de infraestruturas virtualizadas em ambientes de internet das coisas [manuscrito] / Allan Medeiros de Lima. - 2019.

36 p.

Digitado.

Trabalho de Conclusão de Curso (Graduação em Computação) - Universidade Estadual da Paraíba, Centro de Ciências Exatas e Sociais Aplicadas , 2019.

"Orientação : Prof. Dr. Rodrigo Alves Costa , Coordenação do Curso de Computação - CCEA."

1. Internet das Coisas. 2. Virtualização. 3. Redes Definidas por Software. 4. Containers. I. Título

21. ed. CDD 005.1

Elaborada por Lyedja A. M. de Oliveira - CRB - 15/881 BSC7/UEPB

Allan Medeiros de Lima

UMA ABORDAGEM PARA O GERENCIAMENTO DE INFRAESTRUTURAS VIRTUALIZADAS EM AMBIENTES DE INTERNET DAS COISAS

Trabalho de Conclusão de Curso apresentado ao Curso de Bacharelado em Ciências da Computação da Universidade Estadual da Paraíba, em cumprimento à exigência para obtenção do grau de Bacharel em Ciências da Computação.

Aprovado em 18/06/2019

BANCA EXAMINADORA

Prof. Dr. Rodrigo Alves Costa
(Orientador)

Prof. Me. Laudson Silva de Sousa
(Examinador)

Prof. Esp. Fábio Júnior F. da Silva
(Examinador)

RESUMO

Com o aumento do número de dispositivos usados na IoT, surgiu a necessidade de gerenciar estes dispositivos e suas aplicações. Além disso, o gerenciamento de aplicações IoT deve ser realizado de forma apropriada e eficiente para garantir uma performance funcional mais completa. A virtualização baseada em *containers* e as redes definidas por *software* têm sido sugeridos para contornar os problemas de gerenciamento de recursos computacionais e superar os problemas enfrentados pelas infraestruturas de redes atuais. Nesta perspectiva, esta monografia apresenta um estudo de soluções para gerenciamento de infraestruturas virtualizadas no cenário IoT através da utilização de Docker *containers* e redes definidas por *software*, e tem como objetivo proporcionar uma melhor qualidade nos serviços das aplicações IoT que circulam na rede. Para atingir este objetivo foram realizados, em um ambiente controlado de testes, sobrecargas de fluxos de dados entre diversos dispositivos IoT para um servidor MQTT. Além disso, analisamos por meio de ferramentas de monitoramento em tempo real os dados referentes as aplicações IoT, Docker *containers* e também na infraestrutura onde essas aplicações estão hospedadas. Portanto, foi possível observar o custo computacional para a implantação desses serviços através das tecnologias Docker e redes definidas por *software*. Por fim, as tecnologias Docker e redes definidas por *software* mostraram-se capazes de trabalhar de forma flexível e uniforme em ambientes com grande diversidade de dados, promovendo gerenciamento e escalabilidade na rede de forma integrada em ambientes definidos por IoT.

Palavras-chave: Internet das Coisas. Virtualização. Redes definidas por Software. *Containers*.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

API	Application Programming Interface
CNM	Container Networking Model
ISP	Internet Service Provider
IPv6	Internet Protocol Version 6
IoT	Internet of Things
M2M	Machine to Machine
NOS	Network Operating System
OSI	Open Systems Interconnection
OVS	Open Virtual Switch
PIF	Physical Interface
SaaS	Software as a Service
SDN	Software-Defined Networking
QoS	Quality of Service
VIF	Virtual Interface
VM	Virtual Machine
MQTT	Message Queuing Telemetry Transport
VXLAN	Virtual Extensible LAN

SUMÁRIO

1 INTRODUÇÃO

Recentemente, pesquisas apontam que o número de dispositivos de internet das coisas (do inglês, *Internet of Things* – IoT) tende a expandir entre 20 e 46 milhões de dispositivos até o ano de 2020, permitindo que futuramente possamos conectar qualquer dispositivo além dos dispositivos tradicionais de comunicação como *laptops* e *smartphones*, estendendo desta forma para aplicações domésticas como geladeiras, portas de garagem, sistemas industriais para pessoas reais, etc. (GARTNET RESEARCH, 2017).

Nos últimos anos, diversas soluções vêm sendo propostas e implantadas para lidar com o aumento do consumo de dados e o grande número de dispositivos, contudo, tais soluções não foram projetadas para atender bilhões de novos usuários e dispositivos, os quais necessitam associar-se a rede durante um curto período de tempo. Neste contexto, segundo Vilalta et al. (2016), as novas capacidades de programabilidade da rede são providas através da virtualização da rede (do inglês, *network virtualization*) juntamente com as redes definidas por software (do inglês, *software defined networking* – SDN), que surgem como dois habilitadores tecnológicos importantes para o suporte de serviços IoT pelas atuais infraestruturas de rede. Com o uso de redes SDN é possível separar o plano de controle do plano de dados. Conforme Kreutz et al. (2015), essa separação é a chave para prover a flexibilidade desejada nas infraestruturas de redes atuais, dividindo o problema de controle de rede em partes tratáveis, facilitando a criação e introdução de novas abstrações da rede ao passo que simplifica o gerenciamento, evolução e a inovação das redes atuais.

Infraestruturas dirigidas pela virtualização permitirão endereçar grande diversidade de serviços IoT. Deste modo, as redes SDN proveem as capacidades operacionais de gerenciar infraestruturas de redes para explorar novas oportunidades de negócios, como a IoT, contudo, a quantidade massiva de dispositivos IoT proporcionou o surgimento de diversos problemas, não somente estritos a rede, mas também em termos de capacidades de computação e armazenamento (DÍAZ; MARTÍN; RUBIO, 2016). Alguns problemas mais específicos para endereçar os novos ambientes IoT que precisam serem tratados são: (i) interoperabilidade, (ii) eficiência, (iii) gerenciamento e (iv) escalabilidade. Diante disso, a IoT pode tirar proveito da computação em nuvem e da computação de borda para facilitar o provimento de recursos requeridos pelos serviços IoT (KWANG-MAN et al., 2018). A conectividade inteligente e a capacidade de computação são aspectos importantes a serem concedidos, de modo que a rede tradicional seria inadequada para estar envolvida em tal conexão (YASSEIN; ABUEIN; ALASAL, 2017). Por essas razões, a necessidade de uma conectividade completa entre os elementos da IoT é urgente, e consequentemente, a ideia de combinar SDN para alavancar as infraestruturas para o uso mais eficiente da IoT é eminente.

Buscando tratar os problemas mencionados acima, neste trabalho prepõe-se uma abordagem para o gerenciamento de infraestruturas virtualizadas com suporte a implantação de aplicações IoT de forma automatizada. A abordagem proposta visa estender o uso de plataformas de gerenciamento de recursos virtuais através do gerenciamento centralizado da rede. Além disso, esta proposta tem como principais características a implantação de aplicações IoT através do *Docker*, gerenciamento dos recursos computacionais para aplicações IoT através do *Docker Swarm* e por fim o gerenciamento dos recursos virtualizados de rede com um controlador SDN. Neste trabalho os recursos de computação e rede são gerenciados de forma integrada, uma vez que as atuais tecnologias de conteinerização não suportam tal características.

1.1 Motivação

O crescimento da IoT representa um aumento significativo de novos dispositivos para os quais a infraestrutura de *Internet* atual não está preparada. Nesta perspectiva, uma nova arquitetura de rede é requerida para gerenciar a quantidade de fluxos gerados com as aplicações IoT e permitir a existência de serviços diferenciados com requisitos específicos (THUBERT; PALATTELLA; ENGEL, 2015). Iniciativas acadêmicas e industriais tem voltado seus esforços para solucionar grande parte dos problemas nas infraestruturas de redes atuais através das soluções SDN, sendo que implementações SDN já são encontradas nas WANs (do inglês, *Wide Area Network*), ambientes IoT, redes de provedores de acesso a Internet (do inglês, *Internet Service Provider* – ISP), redes de *data centers*, dentre outros (WAN et al., 2016).

Na literatura atual muitas abordagens buscam solucionar os problemas da IoT, no entanto, há uma lacuna de pesquisa entre aspectos teóricos e implementações práticas. Questões como políticas de implantação, problemas com vários serviços específicos de fornecedores e integração de vários dispositivos exigem boa investigação antes da implantação real (BERA; MISRA; VASILAKOS, 2017). Tais aspectos motivaram o desenvolvimento desta pesquisa, no sentido de que existe a necessidade de investigar novos meios para a implantação de aplicações IoT em infraestruturas virtualizadas, fazendo um gerenciamento dos recursos computacionais através de ferramentas de gerenciamento de *containers* juntamente com soluções SDN, buscando ter o controle dos recursos computacionais e de rede de forma coordenada e integrada.

1.2 Objetivos

Como objetivo geral, esta pesquisa visa investigar o uso de soluções baseadas em *containers* para a implantação de serviços e aplicações IoT e a partir disso promover as capacidades de gerenciamento de rede através de redes SDN.

1.2.1 Objetivos Específicos

O desenvolvimento deste trabalho envolve aspectos complementares que definem diferentes áreas de pesquisa, levando assim à elicitação dos seguintes objetivos específicos:

- Revisar o estado da arte sobre a IoT, redes SDN, ferramentas de gerenciamento de *containers*.
- Simular um ambiente virtual de testes, tendo como base os recursos de virtualização computacional e de rede.
- Validar e avaliar a proposta do uso de aplicações IoT através de *containers* e de redes SDN por meio de experimentos específicos.

1.3 Metodologia

Levando em consideração os objetivos elicitados na seção 1.2, um conjunto de medidas foram elaboradas como parte da metodologia deste trabalho. Assim, a realização da mesma pode ser dividida em quatro momentos de referências pertinentes a diferentes métodos e técnicas que foram utilizadas neste trabalho, sendo estes descritos a seguir:

1. Revisão bibliográfica: momento inicial da pesquisa, quando foram organizados e selecionadas as pesquisas que apresentavam as propostas mais recentes no âmbito de IoT, SDN e soluções para o gerenciamento de *containers*;
2. Projeto e desenvolvimento da abordagem proposta: momento em que foram analisadas as principais funcionalidades do Docker, Docker Swarm, Ryu como controlador SDN, Fiware para simular as aplicações IoT. Além disso, tratamos o problema de gerenciamento de recursos de redes através da utilização de redes SDN, haja vista que até o presente momento o Docker não suporta o gerenciamento dos recursos de rede.
3. Validação da pesquisa: momento no qual foram aplicados um conjunto de testes controlados de dados para analisar a eficácia e desempenho da proposta deste trabalho.

1.4 Organização do Trabalho

O restante deste trabalho está organizado como segue: O capítulo 2 fornece a base teórica sobre os principais temas abordados nesta monografia, incluindo a IoT, virtualização e redes SDN, enfatizando as tecnologias utilizadas neste trabalho. Em seguida, o capítulo 3 apresenta a proposta de trabalho e a avaliação. Finalmente, o capítulo 4 fornece as considerações finais.

2 REVISÃO DA LITERATURA

Este capítulo visa tratar dos temas mais importantes que estão relacionados com os principais campos de pesquisas abordados neste trabalho. Para esse fim, inicialmente será abordado sobre IoT, virtualização de *containers* e SDN. Além disso, de forma paralela, são abordadas as principais tecnologias utilizadas neste trabalho.

2.1 Internet das Coisas

A IoT é entendida como uma infraestrutura de rede composta por diversos dispositivos heterogêneos que dependem de tecnologias como dispositivos sensoriais, dispositivos de comunicação, infraestrutura de rede e de processamento de informações necessárias para fornecer serviços avançados destinados a melhorar a qualidade de vida (OJO; ADAMI; GIORDANO, 2016). Segundo Cox et al. (2017), o paradigma IoT pode potencialmente revolucionar a maneira como as pessoas vivem e trabalham por meio de uma infinidade de novos serviços.

A IoT permitirá uma conexão autônoma e segura ao passo que proporciona a troca de dados entre dispositivos e aplicações do mundo real, incorporando inteligência em objetos conectados à *Internet* para prover comunicação, troca de informações e tomada de decisões. Nesta perspectiva, a IoT tende a fornecer serviços surpreendentes que torna em crescente popularidade para as instituições acadêmicas, indústrias bem como governos, uma vez que tem o potencial de trazer benefícios significativos pessoais, profissionais e econômicos (SILVA, 2017b).

Conforme exposto por Alecrim (2017), a IoT desempenha um papel extremamente importante no contexto atual, sendo responsável por ajudar as principais áreas, como saúde, agropecuária, transporte, serviços públicos, dentre outros. A seguir são introduzidos os aspectos relacionados a IoT:

- Hospitais e clínicas: pacientes podem utilizar dispositivos conectados que medem batimentos cardíacos ou pressão sanguínea, por exemplo, e os dados coletados podem ser enviados em tempo real para o sistema que controla os exames;
- Agropecuária: sensores espalhados em plantações podem dar informações mais precisas acerca da temperatura, umidade do solo, probabilidade de chuvas, velocidade do vento e outras informações essenciais para o bom rendimento do plantio. De forma semelhante, sensores conectados aos animais conseguem ajudar de várias maneiras, como por exemplo um chip colocado na orelha de um animal pode fazer o rastreamento, informar o histórico de vacinas e assim por diante;

- Transporte público: usuários podem saber através do *smartphone* ou em telas instaladas nos pontos de parada qual a localização de determinado ônibus. Os sensores também podem ajudar as empresas a descobrirem quando um veículo apresenta defeitos mecânicos e assim saber como está o cumprimento de horários, o que indica a necessidade ou não de reforçar a frota;

- Logística: dados de sensores instalados em caminhões, contêineres e até caixas individuais combinados com informações do trânsito podem ajudar empresas de logística a definir as melhores rotas, escolher os veículos mais adequados para determinada área, quais encomendas distribuir entre a frota ativa e assim por diante;

- Serviços públicos: sensores em lixeiras podem ajudar a administração pública a otimizar a coleta de lixo; já carros podem se conectar a uma central de monitoramento de trânsito para obter a melhor rota para aquele momento, assim como para ajudar o departamento de controle de tráfego a saber quais vias da cidade estão mais movimentadas naquele instante.

2.2 Virtualização

A virtualização surgiu como uma opção para resolver os problemas enfrentados com os antigos *mainframes*, que eram computadores de grande porte com *softwares* desenvolvidos e exclusivos apenas para o seu próprio sistema operacional, pois eles vinham acompanhados de bibliotecas específicas para cada *mainframe*. Quando um software era necessário e este não existia para o sistema operacional da máquina (*mainframe*) que estava sendo usada era necessária a instalação de outro sistema operacional para executá-lo. Desta forma, a virtualização é composta por dois principais componentes, o hospedeiro (*host*) e o hóspede (*guest*), sendo o hospedeiro o sistema operacional instalado na máquina física e o hóspede é o sistema virtualizado que será utilizado no topo do sistema virtual instalado na máquina física. O *hypervisor* é o componente responsável por realizar o intermédio visual entre *host* e *guest* e é por onde o usuário irá usar a máquina virtual (do inglês, *Virtual Machine* – VM), além disso, é também responsável por controlar o uso dos recursos da máquina física (*host*), tanto no quesito espaço quanto no desempenho (BARHAM et al., 2003). Exemplos de softwares para criar máquinas virtuais são: VMware Workstation, VMware Player, VMware vSphere (ESX), Microsoft Hyper-V, Citrix XenServer, RedHat KVM e Oracle VirtualBox (ANDERSON et al., 2005).

A virtualização vem se consolidando como uma ferramenta crucial no apoio à gestão dos ambientes de TI. A adoção de suas soluções possibilitou tratar os recursos computacionais, por vezes subutilizadas, de forma mais eficiente, elevando-os a um nível de compartilhamento onde é possível, por exemplo, fazer com que múltiplos sistemas operacionais executem sobre uma única camada de hardware, ou que um programa possa

ser executado sobre um sistema operacional diferente daquele para o qual foi desenvolvido. Segundo Silva (2017a), a virtualização se mostra como uma ótima solução quando o assunto é redução de gastos, pois com o emprego desta técnica um único computador pode realizar de forma segura os serviços eficientes que para serem ofertados em um ambiente não virtualizado necessitam de vários computadores dedicados.

Além do exposto, a virtualização é a técnica que permite particionar um único Sistema Computacional em vários outros denominados de máquinas virtuais. Cada máquina virtual oferece um ambiente completo muito similar a uma máquina física. Com isso, cada máquina virtual pode ter seu próprio sistema operacional, aplicativos e serviços de rede. É possível ainda interconectar (virtualmente) cada uma dessas máquinas através de interfaces de redes, switches, roteadores e *firewalls* virtuais, além do uso já bastante difundido de VPN (*Virtual Private Networks*). A virtualização pode auxiliar a se trabalhar em um ambiente onde haja uma diversidade de plataformas de *software* (sistemas operacionais) sem ter um aumento no número de plataformas de *hardware* (máquinas físicas). Assim, cada aplicação pode executar em uma máquina virtual própria, possivelmente incluindo suas próprias bibliotecas e seu sistema operacional que, por sua vez, executam em uma plataforma de *hardware* comum (CARISSIMI, 2008).

2.2.1 Virtualização Baseada em *Containers*

A virtualização baseada em *containers* usa o *kernel* do sistema operacional hospedeiro para executar várias instâncias isoladas. Ao fazer uso do *kernel* do sistema operacional hospedeiro e não depender de um *hypervisor* para gerir os recursos, os *containers* tornam-se mais leves do que outras soluções, além de permitir instâncias completamente isoladas. Cada *container* tem seu próprio sistema de arquivos *root*, processos, memória, dispositivos e interface de rede (FERREIRA et al., 2017).

Os *containers* estão preparados para mudar o modo como as aplicações são criadas, enviadas, implantadas e instanciadas. Esta abordagem é capaz de acelerar a entrega de aplicações, facilitando o empacotamento junto com suas dependências. Como resultado, o mesmo aplicativo *conteinerizado* pode operar em diferentes ambientes de desenvolvimento, teste e produção. A plataforma pode ser um servidor físico, servidor virtual, nuvem pública ou dispositivo de rede (LEARN, 2014).

De acordo com Learn (2014) os *containers* são executados em isolamento, compartilhando uma instância do sistema operacional. Essa abordagem pode melhorar a entrega de aplicações de várias maneiras, como por exemplo, redução de custos, aceleramento no desenvolvimento de aplicações, simplificação de segurança, facilitação no processo de adoção de novos modelos de TI através de microsserviço (do inglês, *microservices*).

2.2.1.1 Máquinas Virtuais vs. Contêineres

Containers e máquinas virtuais permitem que vários aplicativos sejam executados nos mesmos sistemas físicos. Eles diferem no grau de atendimento a diferentes tipos de negócios e requisitos de TI (LEARN, 2014).

Uma máquina virtual é uma emulação de um sistema de computador. Simplificando, torna possível executar o que parece ser muitos computadores separados no *hardware* que é na verdade um computador. Cada máquina virtual exige seu próprio sistema operacional subjacente e o *hardware* é virtualizado. Um *hypervisor* é um software que cria e executa máquinas virtuais e está localizado entre o *hardware* e a máquina virtual, sendo necessário para virtualizar o servidor. As máquinas virtuais, no entanto, podem ocupar muitos recursos do sistema. Cada máquina virtual executa não apenas uma cópia completa de um sistema operacional, mas uma cópia virtual de todo o *hardware* que o sistema operacional precisa executar. Isso rapidamente se soma a muitos ciclos de RAM e CPU. Isso ainda é econômico em comparação com a execução de computadores reais separados, mas, para algumas aplicações, pode ser um exagero, o que levou ao desenvolvimento de *containers* (BAUER, 2018).

Já na virtualização baseada em *containers*, a execução e a implantação de aplicações ocorre sem a necessidade da instanciação de um sistema operacional por completo. Os *containers* possuem pacotes e imagens com suas próprias bibliotecas e arquivos que garantem a interoperabilidade entre as diversas requisições das aplicações. Este tipo de virtualização dispensa o uso do *hypervisor*, sendo a camada que gerencia os *containers* se conecta diretamente ao sistema operacional para assim chegar à camada de recursos físicos do *host*. *Containers* são vistos como uma possível sucessão da utilização de máquinas virtuais, apresentando melhores resultados de desempenho quanto à eficiência dos recursos e à flexibilidade no gerenciamento do ambiente (YADAV; SOUSA; CALLOU, 2018).

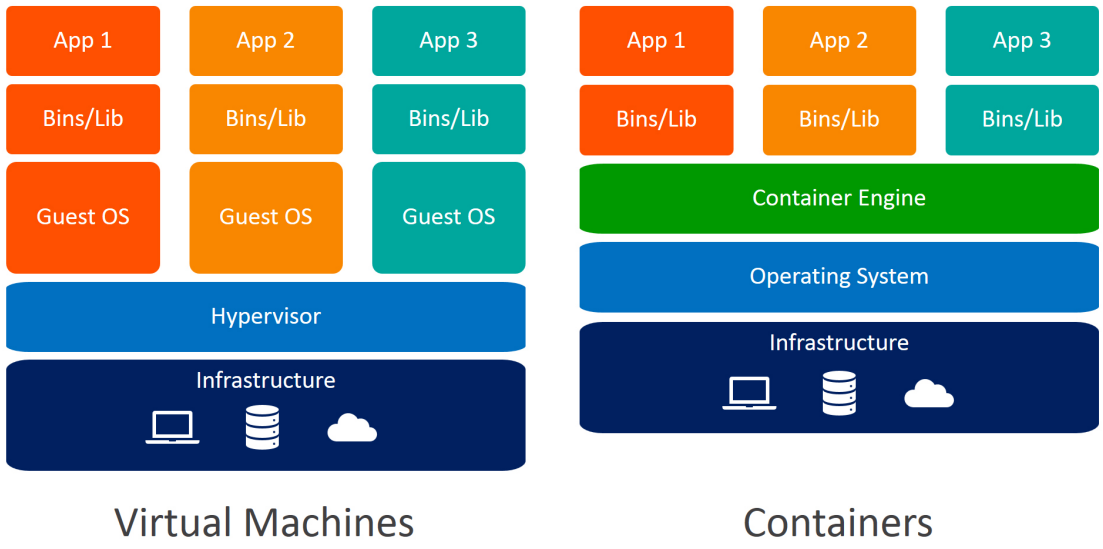
Os *containers* também reduzem a sobrecarga de gerenciamento, compartilhando um sistema operacional comum com apenas um único sistema operacional que precisa de cuidados e alimentação para correções de *bugs*, *patches* e assim por diante. Logo, a abordagem de *containers* são mais leves e mais portáteis que as máquinas virtuais (CHAMBERLAIN, 2018).

De acordo com Yadav, Sousa e Callou (2018) as principais características que tornam a utilização de *containers* mais atrativa são a portabilidade e a rápida entrega de aplicações. Com a portabilidade os *containers* são projetados em uma única unidade que pode ser instanciada em diversos ambientes. Já com a rápida entrega de aplicações o fluxo que os *containers* percorrem são otimizados, garantindo uma rápida entrega de acordo com a necessidade do usuário.

Tanto os *containers* quanto máquinas virtuais têm benefícios e desvantagens,

e a decisão final dependerá de suas necessidades específicas, mas há algumas regras gerais. As máquinas virtuais são uma opção melhor para executar aplicativos que exigem todos os recursos e funcionalidades do sistema operacional, quando é necessário executar vários aplicativos em servidores ou ter uma grande variedade de sistemas operacionais para gerenciar. Os contêineres são uma opção melhor quando sua maior prioridade é maximizar o número de aplicativos em execução em um número mínimo de servidores (BAUER, 2018). A seguir, a Figura 1 ilustra as principais diferenças arquiteturais entre estas duas abordagens de virtualização.

Figura 1 – Comparação de VMs e *Containers*



Fonte: Extraído de (RAZA, 2018).

2.2.2 Docker

O Docker[1] é uma plataforma *Open Source* escrito em GO[2], que é uma linguagem de programação de alto desempenho desenvolvida dentro do Google, que facilita a criação e administração de ambientes isolados. O Docker possibilita o empacotamento de uma aplicação ou ambiente inteiro dentro de um *container*, e a partir disso o ambiente inteiro torna-se portável para qualquer outro *host* que contenha o Docker instalado. Isso reduz drasticamente o tempo de implantação (*deployment*) de alguma infraestrutura ou até mesmo aplicação, pois não há necessidade de ajustes de ambiente para o correto funcionamento do serviço, o ambiente é sempre o mesmo, configure-o uma vez e replique-o quantas vezes quiser (DIEDRICH, 2015).

A demanda por rapidez em execução e economia de recursos computacionais acarretou no crescimento da utilização da plataforma do Docker. A tecnologia por

[1] https://www.docker.com/
[2] https://golang.org/

trás dessa infraestrutura possibilita que uma aplicação seja empacotada juntamente com suas dependências em um *container*, possibilitando a sua execução em qualquer ambiente. Além disso, desenvolvedores e administradores beneficiam-se de uma ambiente confiável e de baixo custo para criar, enviar e executar aplicações distribuídas (MONTEIRO; XAVIER; VALENTE, 2017). O grande uso da tecnologia do Docker deve-se ao fato desta prover diversos benefícios. De acordo com OPSERVICES (2018) a tecnologia Docker apresenta os seguintes benefícios:

- Modularidade: A modularidade permite que o desenvolvedor desabilite uma parte do aplicativo. Dessa forma, podem ser realizadas atualizações de reparo ou até mesmo adição de funcionalidades, sem a necessidade de interromper todo o *software*. Outro ponto é a possibilidade de compartilhar processos entre diferentes aplicativos.
- Camadas e controle de versão de imagens: Um arquivo Docker pode ser formado por diversas camadas diferentes, onde se dividem em dois grupos. O primeiro grupo é denominado como *imagens* que são formadas por diferentes camadas. Com a sua utilização, o usuário pode facilmente compartilhar um aplicativo ou um conjunto de serviços em diversos ambientes. Quando há alguma alteração na imagem, ou uso de um comando como executar ou copiar, é criada uma camada. O segundo grupo são os *containers*, que são formados na reutilização das camadas. Um *container* é o local onde estão as modificações da aplicação que está em execução.
- Reversão: A funcionalidade da reversão permite recuperar a versão anterior de um *container* devido as camadas criadas. O processo se mostra ainda mais eficiente por ser compatível com a abordagem de desenvolvimento ágil. Dessa forma, a equipe pode facilmente contar com as práticas de integração e implantação contínua, sem perder a eficiência no desenvolvimento da aplicação.
- Implantação rápida: Como o tempo e desempenho da implantação ocorrem simultaneamente, uma implantação que levaria horas em outros métodos pode chegar a levar apenas alguns segundos para ser concluída. Para aumentar a eficiência no desenvolvimento de programas, as empresas buscam alternativas, como o Docker. Além de agilizar os processos, a plataforma possibilita ao desenvolvedor a rapidez de acessar uma versão anterior, caso encontre algum problema, trazendo maior produtividade e segurança.

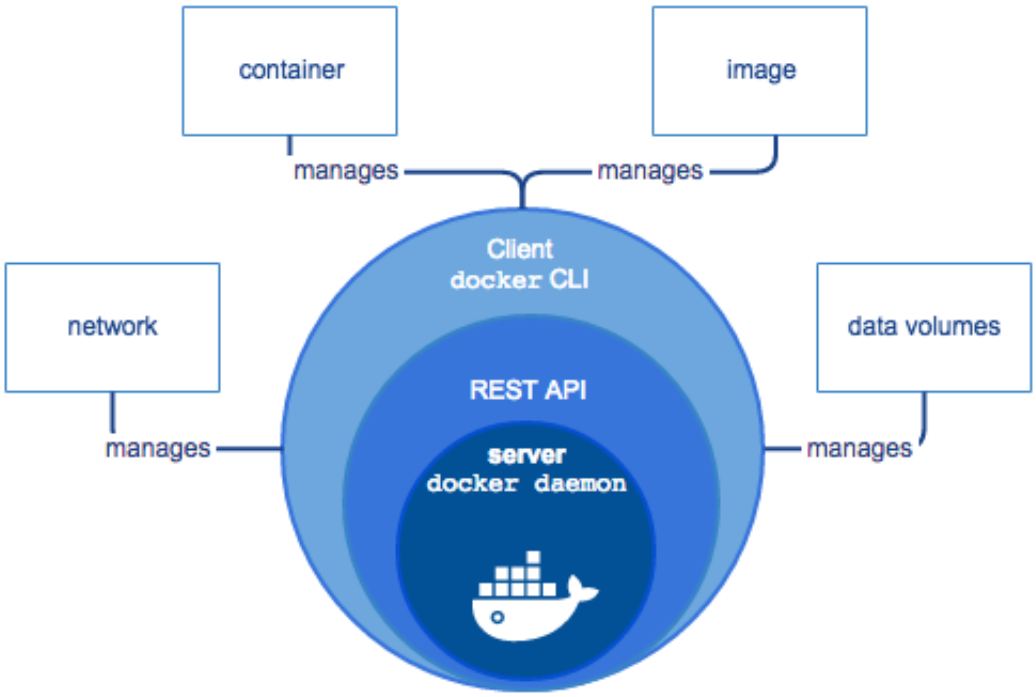
2.2.2.1 Docker Engine

O Docker Engine é a camada de *software* que permite a criação, a execução e a implantação de aplicações com o uso de *containers*. Possibilita o isolamento do ambiente de execução das aplicações em um mesmo *host*, eliminando o *overhead* (sobrecarga). Isso

ocorre porque os recursos do *kernel* (núcleo) do sistema operacional são compartilhados entre *containers* e gerenciados pelo Docker Engine. Com o compartilhamento do *kernel* do *host*, o custo computacional de gerenciamento torna-se menor em relação ao obtido com o uso de máquinas virtuais (GERMANO; GOIÁS; NARCISO, 2018).

O Docker Engine permite que os aplicativos em *containers* sejam executados em qualquer lugar de forma consistente em qualquer infraestrutura, resolvendo o problema da dependência para desenvolvedores e equipes de operações (DOCKER, 2018). Como exposto na figura 2, o Docker Engine é capaz de gerenciar os recursos de rede, os recursos de armazenamento para *containers* (volumes), imagens e *containers* em tempo de execução.

Figura 2 – Docker engine



Fonte: Adaptado de (LIU; ZHAO, 2014).

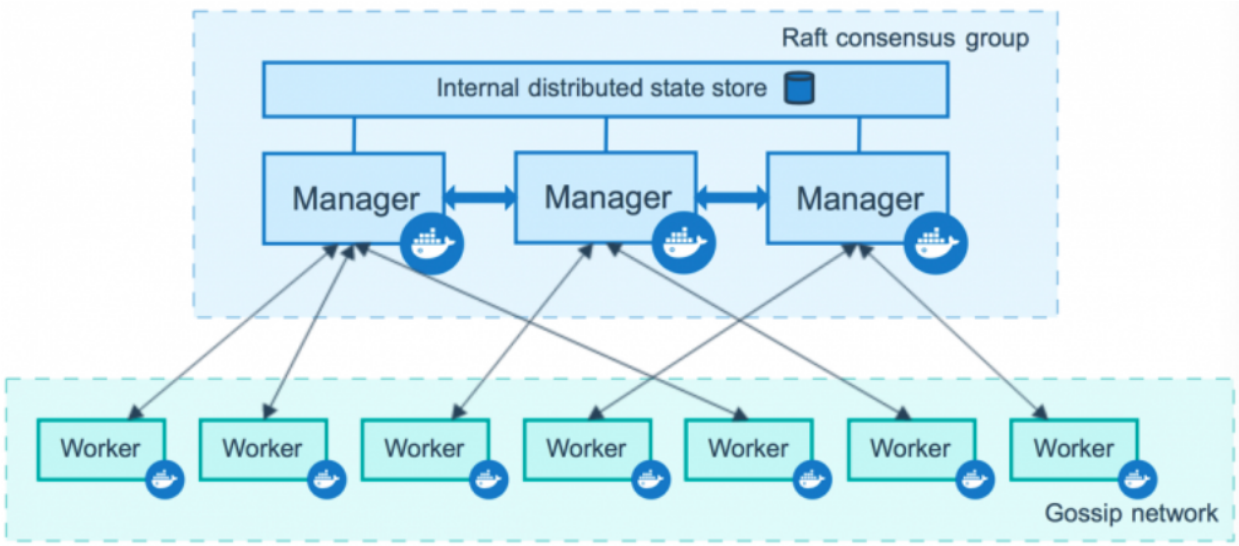
2.2.2.2 Docker Swarm

Docker Swarm é uma ferramenta de orquestração de *containers* nativa do Docker. Proporciona de maneira fácil e organizada a criação de *clusters* e disponibilização de serviços neste ambiente (MONTEIRO; ALMEIDA, 2017). Com o Docker Swarm, os administradores e desenvolvedores de TI podem estabelecer e gerenciar um *cluster* de nós Docker como um único sistema virtual (LUCIO et al., 2017). Segundo Monteiro e Almeida (2017) as principais características do Docker Swarm são:

- Gerenciamento de cluster integrado com Docker Engine: Através do Docker CLI pode-se criar e administrar *cluster*, sendo disponibilizados serviços, não necessitando de software adicional para orquestração de *containers*.
- Adaptação de estado desejado para o *Cluster*: O Docker Swarm monitora constantemente o *cluster*, mantendo o estado conforme configurado inicialmente pelo administrador. Se for criado um serviço para executar 10 (dez) réplicas e uma delas parar, o Docker Swarm automaticamente cria uma nova.
- Segurança: Utiliza o protocolo TLS para comunicação segura entre os nós.

O Docker Swarm transforma um conjunto de hospedeiros em um único hospedeiro virtual maior, chamado de *Swarm*. Há dois tipos de nós no Docker Swarm: nós gerentes (*manager nodes*) e nós trabalhadores (*worker nodes*). Cada nó é uma instância do Docker Engine participante do *Swarm*. Os nós gerentes despacham unidades de trabalho chamadas de tarefas para os nós trabalhadores. Eles também gerenciam o *cluster*, ajudando a manter o estado ideal definido pelo desenvolvedor. Os nós trabalhadores recebem e executam as tarefas. Uma tarefa, uma vez instanciada, não pode mudar de nó. Ela executa no nó designado ou falha. Na terminologia do Docker Swarm, um serviço (*service*) é a definição de uma tarefa. Nela, é definida a imagem de *container* a ser usada e qual comando executar quando a tarefa começar (FILHO; GAMA, 2016). A Figura 3 expõe os conceitos de nós *managers* e *workers* no docker swarm.

Figura 3 – Docker Swarm



Fonte: Adaptado de (NAIK, 2016).

2.2.2.3 Docker Compose

Docker Compose é uma ferramenta nativa do Docker que possibilita a criação e execução de múltiplos *containers*. Se considerarmos que temos em um mesmo projeto um *container* para executar a aplicação, outro para executar o banco de dados, etc. Neste cenário o Docker Compose pode centralizar as configurações desses *containers* em um mesmo arquivo, o *docker-compose.yml*, e criar esses *containers* de forma facilitada. De acordo com Docker (2017) o Docker Compose permite: (i) criar vários ambientes isolados em um único *host*; (ii) preservar dados de volume ao criar *containers*; (iii) recriar *containers* que foram atualizados.

2.2.2.4 Docker *Containers* e Aplicações de Internet das Coisas

Os *containers* oferecem suporte nativo a todos os recursos necessários para executar um aplicativo em dispositivos IoT restritos por recursos. Estes dispositivos são portáteis e eficientes, permitem rápida implantação de aplicativos e escalonamento rápido de aplicativos. Os aplicativos *front-end* da IoT se beneficiariam mais com o modelo de *containers*, pois facilita a implantação, a execução e a atualização dos aplicativos (SINGH, 2017).

A segurança é um dos aspectos mais importantes da IoT. Os aplicativos implantados em *containers* são mais seguros do que os aplicativos implementados no sistema operacional simples. Isso ocorre porque, mesmo que uma segurança de *container* seja comprometida, ela não afetará outros *containers* em execução no sistema operacional do *host*, pois os aplicativos e os usuários são isolados por *container* (SINGH, 2017). De acordo com Tozzi (2018) e Singh (2017) são listados a seguir os principais recursos da abordagem de *containers* que podem ajudar na construção de uma ótima solução de IoT:

- Com o Docker Containers, não é preciso se preocupar em configurar o ambiente base no sistema operacional com as dependências do aplicativo. Os *containers* envolvem o *software* em um sistema de arquivos completo que contém tudo o que é necessário para executar o código em tempo de execução, ferramentas do sistema, bibliotecas do sistema, ou seja, tudo o que pode ser instalado em um sistema operacional de servidor.
- Portabilidade de aplicativos em dispositivos IoT: um aplicativo e todas as suas dependências, como bibliotecas específicas de sensores, podem ser agrupados em um único *container* que é independente da versão do *host* do *kernel* Linux, da distribuição da plataforma ou do modelo de implantação.
- Recursos mínimos de *hardware*. Muitos dispositivos IoT não possuem recursos poderosos de computação e RAM. Sua capacidade de processar atualizações de

software é, portanto, limitada. Os *containers* podem ajudar neste problema porque a instalação de uma nova imagem de um container não requer muito poder de computação. Um dispositivo IoT precisa simplesmente fazer o *download* de uma imagem, colocá-la em qualquer lugar e remover a imagem antiga. O processamento de configuração é mínimo.

- Distribuição geográfica. Em alguns casos de uso, os dispositivos de IoT estão espalhados por uma grande área geográfica. Entregar *software* a estes dispositivos a partir de um único repositório central pode não funcionar bem. Com o Docker, é fácil ativar os registros de imagem em vários locais para atender bem a toda a rede.

- Acesso limitado ou esporádico à rede. Apesar das implicações do termo IoT, nem todos os dispositivos estão bem conectados à *Internet*. Estes dispositivos podem ter largura de banda de rede limitada ou ficar *online* apenas ocasionalmente. O Docker pode ajudar a fornecer atualizações de *software* nessas circunstâncias, porque quando as imagens de um *container* são atualizadas, o Docker baixa apenas as partes da imagem que foram alteradas.

- Ambientes de dispositivos amplamente variáveis. O *software* que é executado em um dispositivo de IoT pode ser quase qualquer coisa. A diversidade de configurações de *software* em dispositivos IoT normalmente dificultaria a instalação de aplicativos, pois os aplicativos teriam que ser configurados para cada tipo de ambiente se instalados por métodos tradicionais.

2.3 Redes Definidas por Software

Apesar da evolução da *Internet*, em termos de aplicações, sua tecnologia, representada pelo modelo *Open Systems Interconnection* (OSI) e pelos protocolos do modelo TCP/IP, não evoluiu o suficiente nas últimas duas décadas. Desde o seu surgimento à *Internet* tornou-se comercial e os equipamentos de rede como roteadores e *switches* tornaram-se "caixas pretas", ou seja, são soluções integradas (os planos de controle e dados são agrupados) verticalmente baseadas em *software* fechado executando em *hardware* proprietário. O resultado desse modelo é o já engessamento da *Internet* mencionado por (ROTHENBERG et al., 2010).

Em contraste a esses problemas enfrentados nas redes atuais surgiu o paradigma de Redes Definidas por Software (do inglês, *Software Defined Networking* – SDN). As redes SDN surgiram em 2005 através de pesquisas realizadas pela Universidade de *Stanford* e é uma nova abordagem de redes de computadores que foi desenvolvida para alterar as limitações da infraestrutura, permitindo aos administradores de rede programaticamente inicializar, controlar, alterar e gerenciar o comportamento da rede dinamicamente

(LAWAL; NURAY, 2018). Este paradigma quebra a integração vertical separando a lógica de controle da rede (o plano de controle) dos roteadores e *switches* subjacentes que encaminham o tráfego (o plano de dados). Além disso, com a separação dos planos de controle e dados, os *switches* de rede tornam-se dispositivos de encaminhamento simples e a lógica de controle é implementada em um controlador logicamente centralizado ou sistema operacional de rede (do inglês, *Network Operating System* – NOS), simplificando a aplicação de políticas e a reconfiguração e evolução da rede (KREUTZ et al., 2015).

A separação da camada de controle (plano de controle) e da camada de infraestrutura (plano de dados) pode ser realizada por meio de uma interface de programação bem definida entre os dispositivos de rede (*switches*) e o controlador SDN. O controlador exerce controle direto sobre o estado nos elementos da camada de infraestrutura por meio dessa interface de programação de aplicações (API) bem definida, conforme ilustrado na figura 4. O exemplo mais notável dessa API é o protocolo *OpenFlow*. Um *switch OpenFlow* possui uma ou mais tabelas de regras de manipulação de pacotes (tabela de fluxo) com cada regra correspondendo a um subconjunto do tráfego e executando ações como descartar, reencaminhar e modificar. Dependendo das regras implementadas pelo controlador, o protocolo *OpenFlow* pode ser usado, por exemplo, pelo controlador para controlar um roteador ou executar outras funções como um balanceador de cargas (KREUTZ et al., 2015).

2.3.1 Características

As redes SDN proporcionam diversas vantagens quando comparadas as redes tradicionais atuais, sendo que suas principais características são tradadas a seguir. De acordo com Tomkis (2018) as principais características da abordagem SDN são:

- **Redes programáveis**: A capacidade de controlar o comportamento da rede usando o *software* externo aos dispositivos de rede física oferece uma maneira fácil de personalizar as redes para suportar novos serviços. Esses serviços podem ser granulares no nível de clientes individuais. Sem as restrições de plataformas fechadas, proprietárias e com o benefício do *hardware* desacoplado do *software*, é possível introduzir novos serviços inovadores e diferenciados rapidamente.
- **Inteligência e controle centralizado**: Com o controle central e inteligência da rede SDN, o gerenciamento de largura de banda, restauração, segurança e políticas tornam-se altamente otimizados. Como resultado tem-se o ganho uma visão holística da rede. A abordagem de rede logicamente centralizada significa que os recursos da rede obtêm o benefício do controle e gerenciamento centrais inteligentes.
- **Interação de rede por meio de *APIs***: com as redes SDN, serviços e aplicativos são separados logicamente do *hardware* e das conexões da rede física. Em vez de

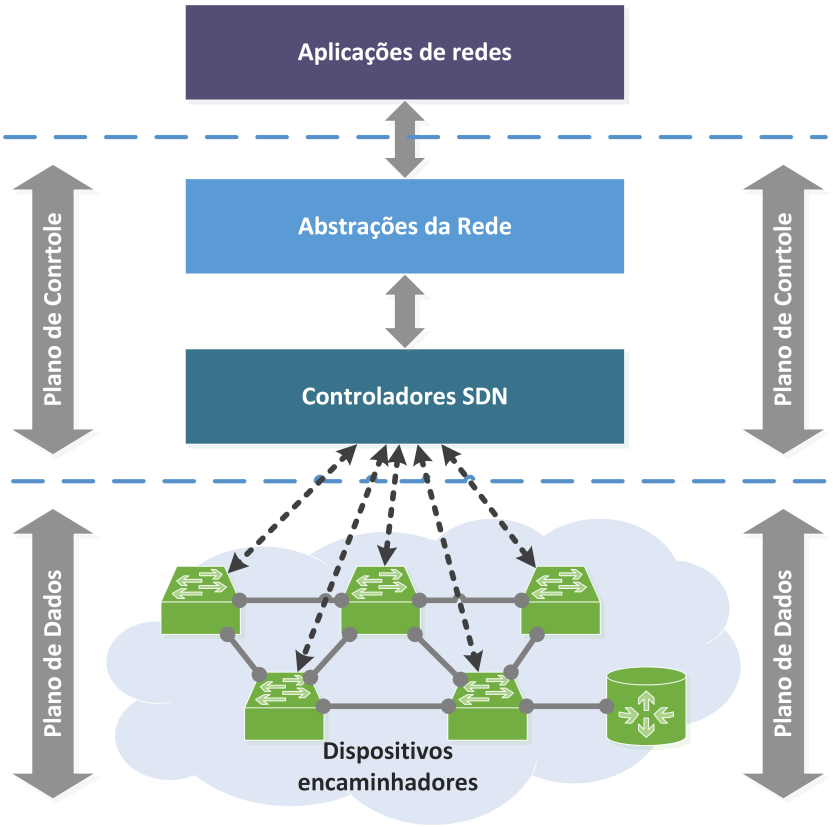
usar conexões físicas e interfaces de gerenciamento que são fortemente acopladas ao *hardware*, aplicativos se conectam através da infraestrutura de rede de forma flexível com o uso de *APIs*.

- **Arquiteturas neutras de fornecedor**: Interoperabilidade e abertura são marcas registradas das arquiteturas SDN. A interoperabilidade de vários fornecedores promove um ecossistema neutro de fornecedor. O uso de *APIs* abertas permite uma variedade de aplicativos, como orquestração de nuvem, software como serviço (do inglês, *Software as a Service* - SaaS) e aplicativos em redes críticas para os negócios. O *software* inteligente das redes SDN pode controlar o *hardware* de diferentes fornecedores que utilizam interfaces programáticas abertas como o *OpenFlow*.

2.3.2 Arquitetura

A arquitetura das redes definidas por software são compostas por três planos, o plano de aplicação, plano de controle e plano de dados. No plano de controle fica o componente chave, o controlador, responsável por receber instruções do plano de aplicação e aplicar as mudanças necessárias no plano de dados. No plano de aplicação ficam os softwares responsáveis pelo gerenciamento da rede. No plano de dados fica o hardware necessário para o encaminhamento de dados (HENRIQUE, 2018). Uma visão simplificada da arquitetura SDN é mostrada na figura 4. A seguir são explorados os principais componentes da arquitetura SDN.

Figura 4 – Arquitetura SDN



Fonte: Adaptado de (KREUTZ et al., 2015).

2.3.3 Uso de SDN para suportar dispositivos IoT

As arquiteturas tradicionais e protocolos de rede para dispositivos IoT não são projetados para oferecer suporte a alto nível de escalabilidade, grande quantidade de tráfego e mobilidade. Estas arquiteturas são ineficientes e têm limitações para atender a esses novos requisitos. Além disso, com o grande número de objetos conectados a rede é difícil gerenciá-los, gerando uma quantidade imensa de dados como um todo, sem ter elasticidade e flexibilidade inerentemente definidas na rede. Se as redes não estiverem preparadas, a inundação de IoT, onde muito tráfego é gerado, pode deixar a rede paralisada (OJO; ADAMI; GIORDANO, 2016).

A interoperabilidade dos dispositivos de IoT pode ser resolvida usando SDN, superando a heterogeneidade de dispositivos. O desafio de interoperabilidade em IoT surge quando temos dispositivos heterogêneos trocando formatos de dados e protocolos diversos para troca de dados máquina a máquina (M2M) e também com a interconectividade de grande número de dispositivos diferentes, sendo que há falta de cooperação e incompatibilidade de capacidade entre dispositivos que podem dificultar o desempenho da rede. No entanto, a abordagem SDN traz flexibilidade que pode ser usada para permitir que diferentes objetos conectados a redes heterogêneas se comuniquem uns

com os outros. Isso será capaz de lidar com conexões simultâneas de várias tecnologias de comunicação. As decisões de gerenciamento de rede, como roteamento, agendamento, podem ser feitas no controlador SDN e, além disso, a capacidade de programação permite atualizações para novas propostas ou até mesmo abordagens de estado limpo (OJO; ADAMI; GIORDANO, 2016).

De acordo com Sakya (2018), para lidar com esse fluxo de dispositivos IoT muitas empresas estão recorrendo à tecnologia SDN. A SDN se tornou uma opção popular para empresas que desejam gerenciar suas redes usando uma solução de *software* mais econômica, especialmente quando comparada com métodos de *hardware* mais caros. A abordagem SDN permite que as empresas otimizem suas redes especificamente para IoT. As redes SDNs podem ajudar com o tráfego da IoT no sentido de: (i) **Realizar o encadeamento de serviços**, permitindo que uma rede SDN crie uma cadeia virtual de serviços, como *firewalls*, proteção contra invasões e VPNs. Uma cadeia de serviços pode então ser definida para lidar com tráfego específico para dispositivos ou serviços específicos, como um dispositivo IoT; (ii) **Através do gerenciamento dinâmico de carga**, permitindo que os administradores de rede ajustem automaticamente as regras de rede, dependendo da carga geral do tráfego de rede; e (iii) **Com o uso de calibração de largura de banda**, sendo que os dispositivos IoT não precisam de acesso contínuo a uma rede. Os dispositivos de IoT precisam apenas de acesso à rede em intervalos específicos, que o calendário de largura de banda permitirá que um administrador de rede leve em consideração.

3 PROPOSTA DE TRABALHO E AVALIAÇÃO

Este capítulo introduz a abordagem proposta neste trabalho, identificando as principais características impostas e abrangendo como são realizadas as configurações e implantações dos serviços utilizados. Além disso, este capítulo está organizado de forma a conter uma visão geral da proposta, uma descrição detalhada do projeto e implementação da solução e por fim o cenário de avaliação.

3.1 Proposta de Trabalho

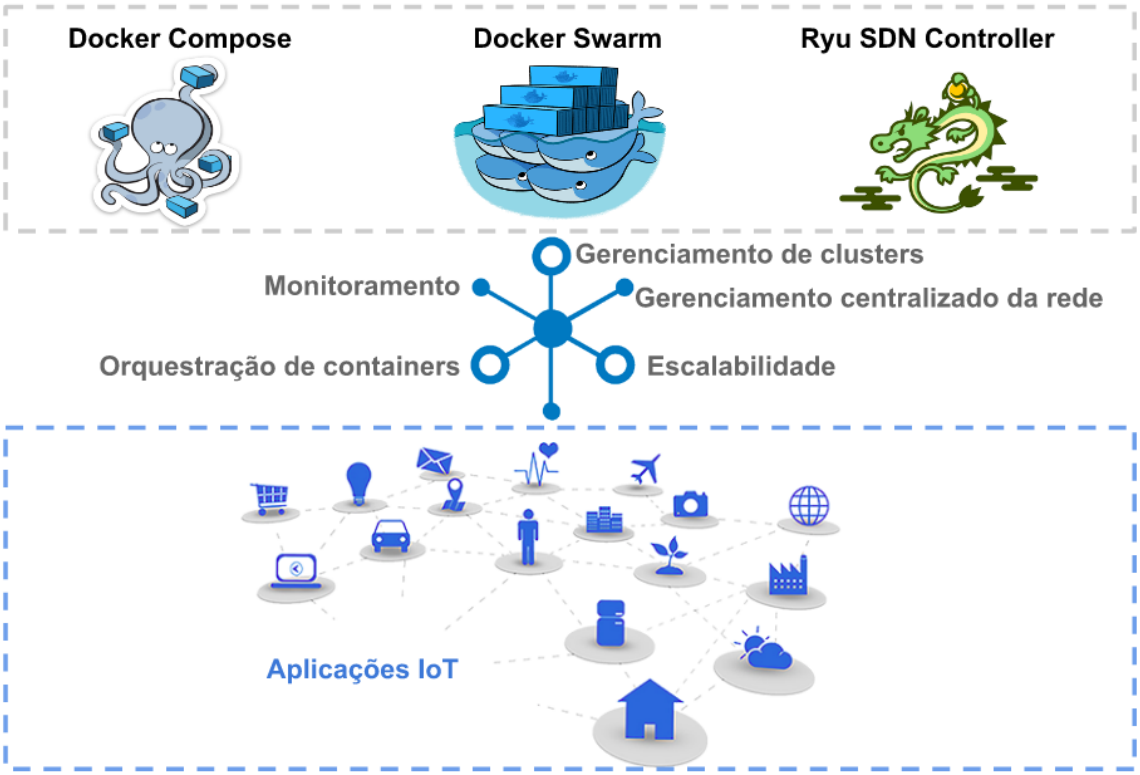
No decorrer do trabalho foi utilizado o Docker para a implantação de aplicações IoT. Assim, as aplicações IoT poderão tirar proveito dos principais benefícios fornecidos por Docker, como a portabilidade, segurança e escalonamento. Dessa forma, o trabalho foca no estudo da utilização do Docker e de SDN no cenário das aplicações IoT.

No contexto do trabalho foram utilizados as ferramentas nativas do Docker (Docker Compose e Docker Swarm) para o gerenciamento, monitoramento e orquestração das aplicações de IoT. As ferramentas utilizadas são ideais para trabalhar no cenário de IoT pois estas soluções possibilitam prover a facilidade no gerenciamento das aplicações quando implantadas na forma de *containers*. No cenário de rede será explorado o paradigma de SDN com o objetivo de trabalhar de forma flexível com a grande quantidade de dados que estará sendo gerado na rede pelas aplicações IoT, visando obter benefícios para melhoria da qualidade dos serviços das aplicações IoT que trafegam na rede. Logo, por meio da utilização de SDN as aplicações IoT visam usufruir dos recursos diferenciados oferecidos por essa tecnologia, como flexibilidade, escalabilidade e gerenciamento centralizado com finalidade de alcançar melhor desempenho na rede.

3.1.1 Projeto e Implementação

Para realizar a implantação da plataforma Mosquitto e dos dispositivos IoT através do FIWARE utilizamos o Docker, pois ambos serviços são disponibilizados na plataforma do Docker através de imagens. Além disso, o Docker Swarm foi utilizado para realizar a implantação dos componentes de monitoramento e do servidor e respectivos clientes MQTT de forma automatizada. Por fim, os recursos SDN são providos pelo OVS e gerenciados pelo controlador SDN Ryu. A visão geral destes componentes e suas interconexões são ilustradas na figura 5.

Figura 5 – Visão geral da solução



Fonte: (Lima, 2019).

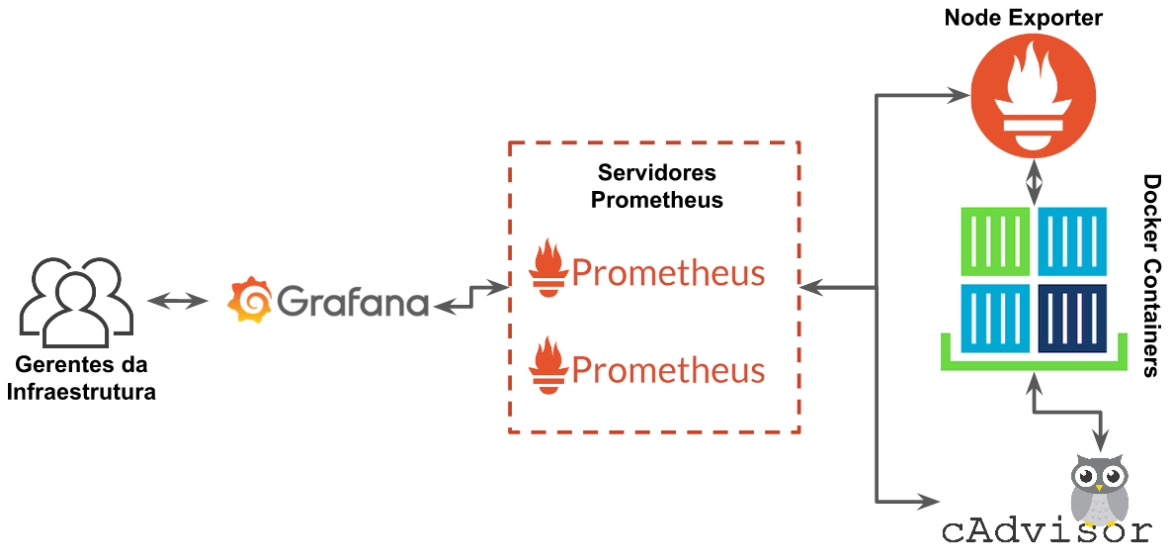
3.1.2 Sistema de Monitoramento

Durante o desenvolvimento do trabalho foram utilizadas um conjunto de ferramentas de monitoramento para darem suporte a tomada de decisões de gerenciamento e orquestração de *containers*. A solução de monitoramento abordada neste trabalho inclui as tecnologias Grafana, Prometheus, Node Exporter e cAdvisor, sendo possível coletar informações acerca da infraestrutura onde estão hospedados os *containers* e também dos próprios *containers* para facilitar as tarefas dos gerentes encarregados por essa infraestrutura. O monitoramento é uma parte essencial de ambientes baseados em Docker, sendo que os principais pontos para se manter uma solução de monitoramento são: (i) detecção de problemas em um estágio inicial para evitar riscos em um nível de produção; (ii) mudanças e atualizações podem ser feitas com segurança, à medida que todo o ambiente é rastreado; e (iii) provê capacidades para que aplicações possam obter melhor desempenho e robustez.

O Prometheus é uma solução de monitoramento que coleta métricas de diferentes formas. O Prometheus é encarregado por armazenar e realizar o monitoramento dos dados referentes aos *containers* que foram selecionados pelo cAdvisor e também da infraestrutura através do Node Exporter. Dessa forma, esses dados colhidos poderão ser consultados pelo Grafana que será responsável por exibir essas métricas de forma gráfica para os gerentes, assim estes poderão ter uma visão geral tanto da infraestrutura como dos

próprios *containers*. Já o Grafana é uma plataforma aberta de monitoramento e *analytics* que ajuda a visualizar os dados coletados pelo Prometheus. A arquitetura da *stack* de monitoramento utilizada neste trabalho pode ser compreendida na Figura 6, onde todos os componentes arquiteturais de monitoramento são expostos bem como suas interligações.

Figura 6 – *Stack* de Monitoramento



Fonte: (Lima, 2019).

Todos os serviços de monitoramento podem facilmente serem integrados ao Docker Swarm, não havendo necessidade de instalar uma camada extra de *software* nos servidores que fazem parte de um *cluster* no Docker Swarm. Pode-se simplificar a inicialização dos dois serviços através de um arquivo yml que define a *stack* de monitoramento. Após definir a *stack* de monitoramento no arquivo denominado *docker-stack.yml* é possível iniciar a implantação destes serviços através do comando *docker stack deploy -c docker-stack.yml monitoring*, sendo o serviço *stack* do Docker utilizado para realizar a implantação dos serviços. A seguir explicamos de forma mais detalhada o arquivo yml utilizado para realizar a implantação dos serviços de monitoramento.

A primeira parte do arquivo está relacionada as configurações básicas de interpretação deste arquivo pelo serviço de stack do Docker. As informações iniciais definem a versão do arquivo yml a ser interpretada pelo Docker bem como a criação de *volumes*, ou seja, os endereços de armazenamento utilizados para guardar as informações de monitoramento tanto do Prometheus quanto do Grafana.

```
1 version: '3.4'
2
3 volumes:
```

```
4     prometheus_data: {}
5     grafana_data: {}
```

A seguir são descritos detalhadamente cada serviço que compõe a *stack* de monitoramento. O primeiro serviço do *docker-stack.yml* é o Prometheus que importa um arquivo prometheus.yml contendo as configurações de diretório do *host* e expõe uma *API* na porta 9090 que é pública e acessível.

```
1  services:
2      image: prom/prometheus
3          volumes:
4            - ./prometheus/:/etc/prometheus/
5            - prometheus_data:/prometheus
6          command:
7            - '--config.file=/etc/prometheus/prometheus.yml'
8            - '--storage.tsdb.path=/prometheus'
9            - '--web.console.libraries=/usr/share/prometheus'
10           - '--web.console.templates=/usr/share/prometheus/consoles'
11         ports:
12           - 9090:9090
```

O segundo serviço é o Node-Exporter. Este serviço é uma imagem Docker provida pelo Prometheus para expor as métricas de recursos como RAM, CPU, armazenamento e rede de um servidor Docker ou de diversos servidores que fazem parte de um *cluster*.

```
1  services:
2      node-exporter:
3          image: prom/node-exporter
4          volumes:
5            - /proc:/host/proc:ro
6            - /sys:/host/sys:ro
7            - /:/rootfs:ro
8          command:
9            - '--path.procfs=/host/proc'
10           - '--path.sysfs=/host/sys'
11           - --collector.filesystem.ignored-mount-points
12           - "^/(sys|proc|dev|host|etc|rootfs/var/lib/docker/containers|
13           rootfs/var/lib/docker/overlay2|rootfs/run/docker/netns|
14           rootfs/var/lib/docker/aufs)($$|/)"
```

```
15          ports:
16            - 9100:9100
```

O terceiro serviço é o cAdvisor que também funciona como um coletor de métricas, porém provê as métricas sobre os Docker *containers*. O serviço do cAdvisor é mantido pelo Google e também está disponibilizado em forma de imagem na plataforma do Docker. Além disso, este serviço pode ser utilizado num *cluster* completo, coletando métricas de todos os recursos disponíveis em diversos servidores. As métricas de monitoramento de *containers* são expostas através de *API* na porta 9100 e são acessíveis de forma flexível o suficiente para serem consultadas frequentemente.

```
1  services:
2      cadvisor:
3          image: google/cadvisor
4          volumes:
5            - /:/rootfs:ro
6            - /var/run:/var/run:rw
7            - /sys:/sys:ro
8            - /var/lib/docker/:/var/lib/docker:ro
9          ports:
10           - 8080:8080
```

Por fim, o Grafana é o último serviço a ser implantado, sendo responsável por coletar as métricas do Prometheus e prover uma *dashboard* gráfica. O serviço do Grafana é acessível na porta 3000.

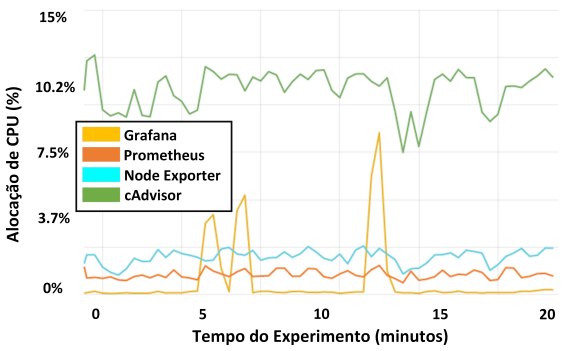
```
1  services:
2      grafana:
3          image: grafana/grafana
4          depends_on:
5            - prometheus
6          ports:
7            - 3000:3000
8          volumes:
9            - grafana_data:/var/lib/grafana
10           - ./grafana/provisioning/:/etc/grafana/provisioning/
11         env_file:
12           - ./grafana/config.monitoring
```

3.2 Avaliação

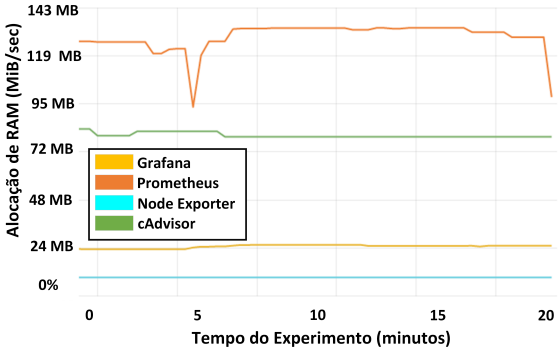
Esta seção detalha a avaliação da solução abordada neste trabalho, tanto a parte de monitoramento quanto da parte de gerenciamento e orquestração de recursos computacionais pelo Docker Swarm e pelo controlador SDN Ryu. Além disso, descrevemos de forma detalhada como a implantação do *testbed* foi realizada. Por fim, analisamos os resultados preliminares obtidos com o desenvolvimento desta pesquisa.

3.2.1 Avaliação do Sistema de Monitoramento

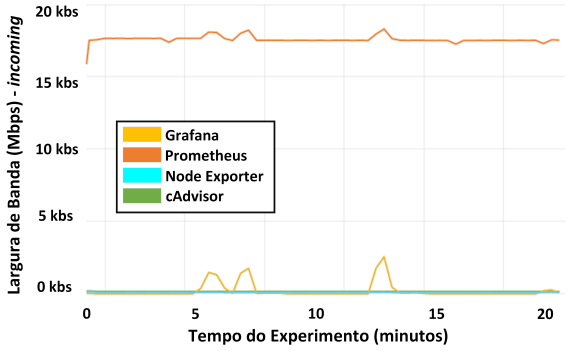
Após realizar as configurações necessárias e a implantação dos serviços foi possível monitorar todos os *containers* em tempo real bem como buscar o histórico desses dados. A figura 7 ilustra o custo dos recursos computacionais e de rede para manter uma solução de monitoramento flexível. Além disso, os recursos de CPU, RAM, rede (*incoming* e *outcoming*) são tratados na figura 7.



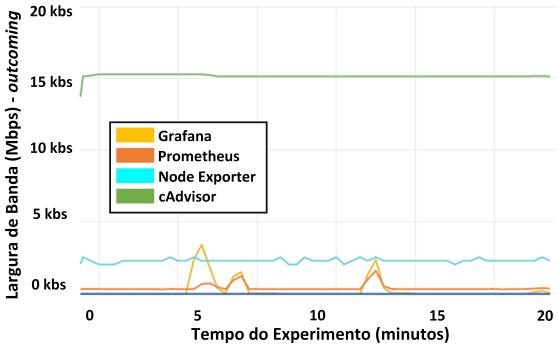
(a) Impacto nos recursos de CPU



(b) Impacto nos recursos de RAM



(c) Impacto na Largura de banda (*incoming*)



(d) Impacto na Largura de banda (*outcoming*)

Figura 7 – Consumo de recursos pela *stack* de monitoramento

A solução de monitoramento utilizada neste trabalho mostra-se flexível o suficiente para ser implantada em alta escalabilidade em ambientes IoT. Além disso, não requer grandes esforços técnicos durante as etapas de configuração. Por fim, foi possível identificar

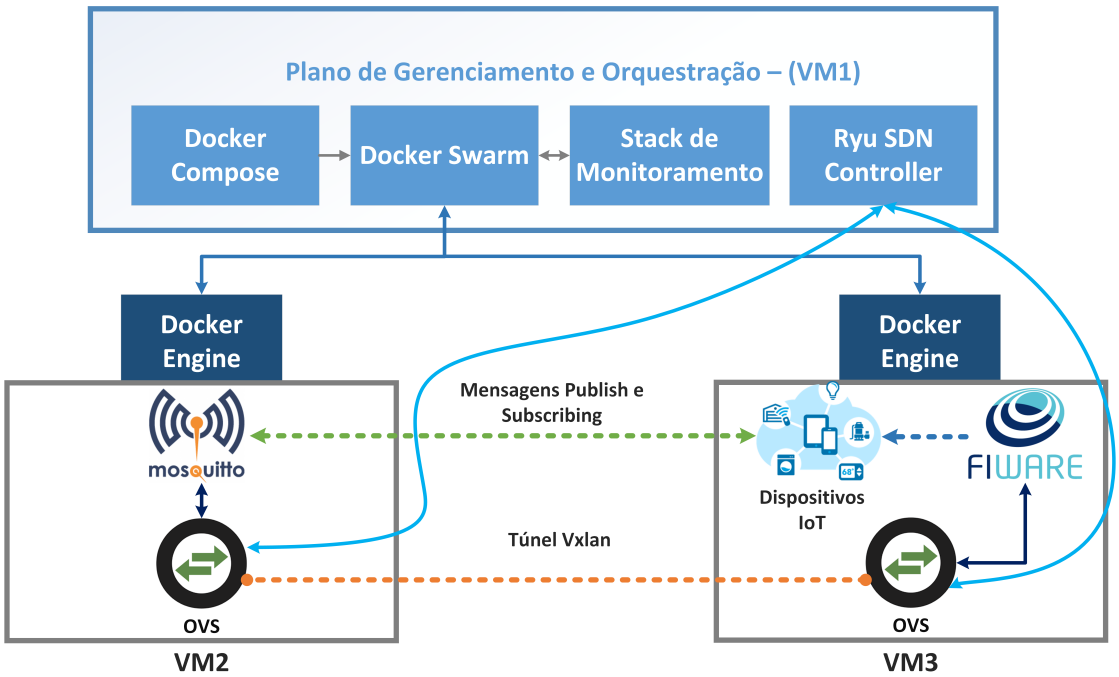
que esta solução consegue abranger um ambiente complexo de IoT, podendo envolver o monitoramento de aplicações IoT que sejam implantadas na forma de *containers* Docker.

3.2.2 Configuração dos Experimentos

O *testbed* é composto por três máquinas virtuais que foram implantadas através do software VirtualBox com sistema operacional Ubuntu Server versão 18.04 LTS. Uma das máquinas virtuais hospeda os componentes de monitoramento. A segunda máquina virtual hospeda o servidor MQTT através da plataforma Mosquitto. A terceira máquina virtual hospeda os 500 dispositivos IoT através do FIWARE, ou seja, os clientes MQTT. Tanto o servidor quando os clientes MQTT foram implantados através de *containers* Docker. Além disso, para conectar os clientes MQTT ao servidor MQTT entre as duas máquinas virtuais foi utilizada uma rede SDN com limitações de largura de banda de 25 Mbps. Todos os 500 clientes MQTT realizam operações de *publishing* e *subscribing* no servidor Mosquitto visando estressar este serviço. A Figura 8 mostra como estão organizados os componentes tecnológicos do testbed.

O uso de um *testbed* abordado neste trabalho visa principalmente provar princípios funcionais básicos da solução proposta, reunindo todos os componentes que envolvem a arquitetura funcional apresentada na figura 8. O objetivo dos testes consistem em prover o gerenciamento dos recursos de rede e computação de forma unificada em aplicações IoT.

Figura 8 – Configuração dos experimentos no testbed



Fonte: (Lima, 2019).

O Mosquitto é uma plataforma *open source* que realiza um serviço de interpretação de mensagens com o protocolo MQTT nas versões 5.0, 3.1.1 e 3.1. O Mosquitto é uma ferramenta leve e adequada para o uso de dispositivos com baixo consumo de energia, computadores e servidores mais poderosos. O protocolo MQTT provê método leve para lidar com mensagens através do modelo *publish/subscribing*. Tal fato faz do Mosquitto uma ferramenta adequada para mensagens de dispositivos IoT tais como sensores de baixa potência ou dispositivos móveis, telefones, computadores embarcados ou microcontroladores. Além disso, o fundação Eclipse (mantém o projeto Mosquitto) disponibiliza a implantação do serviço do Mosquitto através de *containers* Docker. Dessa forma optamos por utilizar o Mosquitto em forma de *container* para facilitar o modo de implantação deste serviço. A seguir apresentamos o arquivo yml utilizado para realizar a implantação do Mosquitto.

```
1 services:
2     mqtt:
3       image: toke/mosquitto
4       networks:
5         - default
6       ports:
7         - 1883:1883
8       restart: unless-stopped
```

Já o FIWARE também é uma plataforma *open source* para o futuro digital. As principais características desta plataforma consistem em (i) conduzir padrões-chave para romper os silos de informação; (ii) Prover facilidade para IoT; (iii) Transformar Big Data em conhecimento; (iv) Habilitar a economia de dados; (v) Desencadear o potencial de dados abertos no momento certo. Além disso, o FIWARE define um conjunto universal de padrões para gerenciamento de dados de contexto que facilitam o desenvolvimento de soluções inteligentes para diferentes domínios, como *Smart Cities*, *Smart Industry*, *Smart Agoodood* e *Smart Energy*.

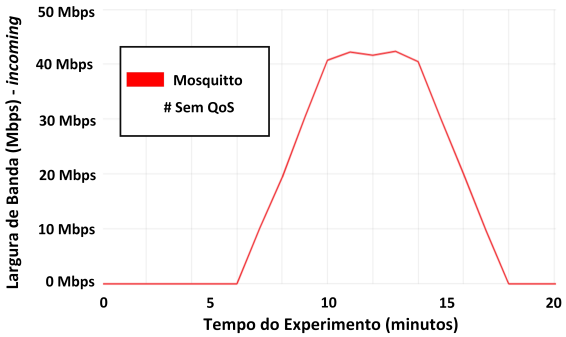
A plataforma FIWARE comporta vários tipos de habilitadores genéricos (do inglês, *Generic Enablers*) e habilitadores específicos de domínios (do inglês, Domain Specific Enablers). Desta forma, a principal ideia inovadora por trás do FIWARE é que objetos ("things") IoT são expostos aos desenvolvedores que utilizam a plataforma FIWARE da mesma forma que protocolos NGSI são expostos. Através disso os desenvolvedores não terão que se preocupar com complexidades variadas de tecnologias, fazendo com que o aprendizado e o uso de *APIs* específicas possam ser usadas e representam todo o contexto de informação abstrato de dados. A plataforma FIWARE também disponibiliza a implantação dos seus serviços através do Docker. Diante de tal fato também optamos por utilizar o serviço do FIWARE através do Docker, pois assim temos acesso as métricas

de monitoramento através da *stack* utilizada e também o gerenciamento e orquestração de recursos através do Docker Swarm e do Ryu. A seguir apresentamos o template utilizado.

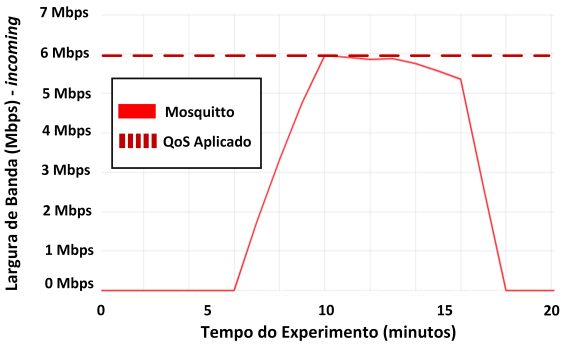
```
services:
    iot-agent:
        image: fiware/iotagent-ul:latest
        networks:
            - default
        ports:
            - "4041:4041"
        environment:
            - "IOTA_CB_HOST=orion"
            - "IOTA_CB_PORT=1026"
            - "IOTA_NORTH_PORT=4041"
            - "IOTA_REGISTRY_TYPE=mongodb"
            - "IOTA_LOG_LEVEL=DEBUG"
            - "IOTA_TIMESTAMP=true"
            - "IOTA_MONGO_HOST=mongo-db"
            - "IOTA_MONGO_PORT=27017"
            - "IOTA_MONGO_DB=iotagentul"
            - "IOTA_PROVIDER_URL=http://iot-agent:4041"
            - "IOTA_MQTT_HOST=mosquitto"
            - "IOTA_MQTT_PORT=1883"
```

3.2.3 Resultados

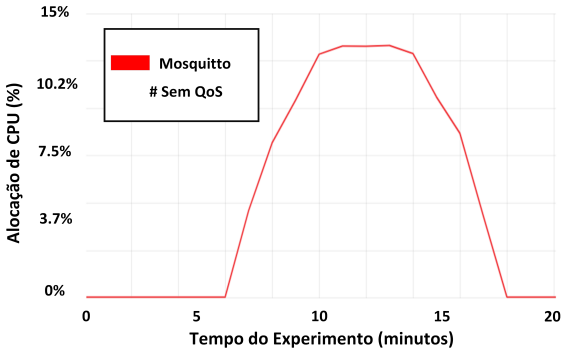
Nesta seção abordamos os resultados adquiridos durante o uso da abordagem de QoS provida pela solução de SDN. A figura 9 contém os resultados obtidos durante os dois experimentos realizados, sendo os recursos de largura de banda, CPU e RAM analisados respectivamente. No primeiro experimento não foi utilizado nenhum recurso de SDN, exceto a conectividade através do *switch* OVS. Desta forma, não é possível ter um controle mais preciso sobre o gerenciamento dos recursos de rede utilizando-se somente a ferramenta do Docker. Já no segundo experimento, utilizamos os recursos de QoS providos pelo controlador SDN Ryu, limitando assim os recursos de largura de banda do servidor MQTT para atender demandas específicas de recursos. Ainda de acordo com a figura 9, o uso de QoS no segundo experimento permitiu observarmos que o impacto na limitação dos recursos de largura de banda, através do controlador SDN Ryu, tem efeitos significativos e diretos nos recursos de CPU e RAM, haja vista que uma vez que poucos dados são trafegados, posteriormente menos processamento é requerido.



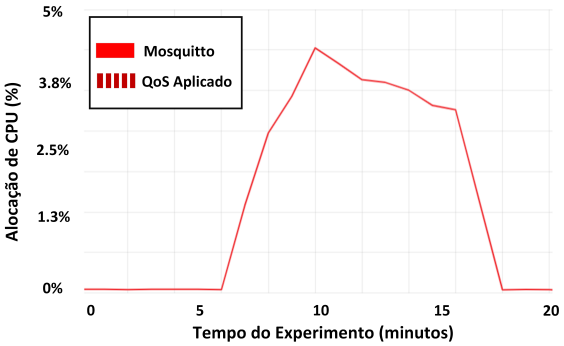
(a) Impacto nos recursos de rede



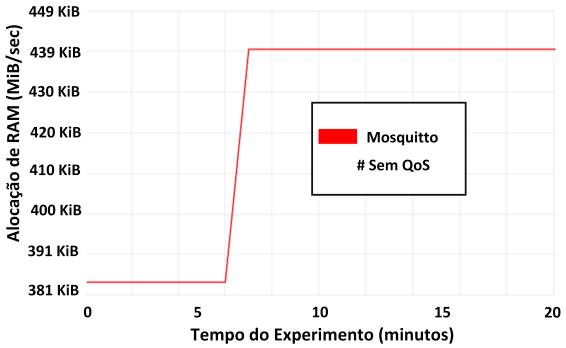
(b) Impacto nos recursos de rede QoS



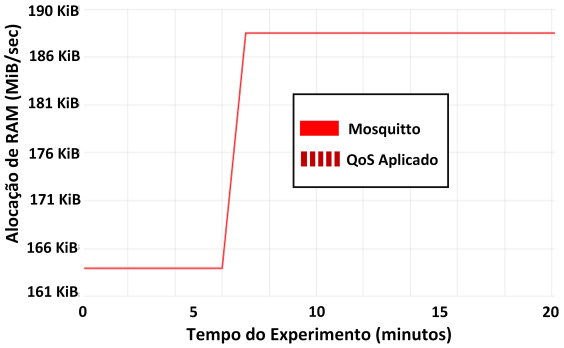
(c) Impacto nos recursos de CPU



(d) Impacto nos recursos de CPU (QoS)



(e) Impacto nos recursos de RAM



(f) Impacto nos recursos de RAM (QoS)

Figura 9 – Utilização de recursos sem e com QoS

Durante a execução deste trabalho, observou-se que o Docker é uma ferramenta na qual soluções IoT podem basear-se para atender requisitos específicos deste cenário. Contudo, esta ferramenta não provê o gerenciamento de recursos de rede, não cobrindo todo o escopo de gerenciamento de uma infraestrutura virtualizada que abarca aplicações IoT. Através das grandes capacidades de gerenciamento e controle da rede pelas redes SDN foi possível gerenciar os recursos de rede, promovendo maior flexibilidade ao usar o Docker juntamente com o controlador Ryu.

4 CONSIDERAÇÕES FINAIS

O desenvolvimento deste trabalho possibilitou uma visão de como o gerenciamento de infraestruturas virtualizadas no contexto da IoT pode ser beneficiada por meio da utilização do Docker juntamente com SDN, sendo que aplicações IoT beneficiam-se dos principais recursos fornecidos por essas tecnologias.

Diante dos testes realizados nos ambientes propostos, observou-se os resultados relacionados aos custos computacionais das aplicações e ambientes, e foi possível notar o quão eficiente e econômico em termos de custos computacionais se torna a implantação de aplicações e serviços IoT por meio da utilização das tecnologias de redes definidas por *software* e Docker *containers*.

Por fim, foi possível observar que a utilização das tecnologias SDN e Docker *containers*, apresentaram resultados bastante significativos nos testes realizados e mostraram habilitadas para lidar com grande quantidade de dados de forma ágil em ambientes de IoT. Através deste estudo foi possível aprofundar-se nas soluções atualmente utilizadas para prover o gerenciamento de infraestruturas IoT.

REFERÊNCIAS

ALECRIM, E. **O que é Internet das Coisas (Internet of Things)?** 2017. Disponível em: <https://www.infowester.com/iot.php>. Acesso em: 09 sep. 2018. Citado na página 11.

ANDERSON, T. et al. Overcoming the internet impasse through virtualization. **Computer**, IEEE, n. 4, p. 34–41, 2005. Citado na página 12.

BARHAM, P. et al. Xen and the art of virtualization. In: ACM. **ACM SIGOPS operating systems review**. [S.l.], 2003. v. 37, n. 5, p. 164–177. Citado na página 12.

BAUER, R. **What's the Diff: VMs vs Containers**. 2018. Disponível em: <https://www.backblaze.com/blog/vm-vs-containers/>. Acesso em: 14 jan. 2019. Citado nas páginas 14 e 15.

BERA, S.; MISRA, S.; VASILAKOS, A. V. Software-defined networking for internet of things: A survey. **IEEE Internet of Things Journal**, IEEE, v. 4, n. 6, p. 1994–2008, 2017. Citado na página 9.

CARISSIMI, A. Virtualização: da teoria a soluções. **Minicursos do Simpósio Brasileiro de Redes de Computadores–SBRC**, v. 2008, p. 173–207, 2008. Citado na página 13.

CHAMBERLAIN, D. **Containers vs. Virtual Machines (VMs): What's the Difference?** 2018. Disponível em: <https://blog.netapp.com/blogs/containers-vs-vms/>. Acesso em: 24 jan. 2019. Citado na página 14.

COX, J. H. et al. Advancing software-defined networks: A survey. **IEEE Access**, IEEE, v. 5, p. 25487–25526, 2017. Citado na página 11.

DÍAZ, M.; MARTÍN, C.; RUBIO, B. State-of-the-art, challenges, and open issues in the integration of internet of things and cloud computing. **Journal of Network and Computer applications**, Elsevier, v. 67, p. 99–117, 2016. Citado na página 8.

DIEDRICH, C. **Docker**. 2015. Disponível em: <https://www.mundodocker.com.br/>. Acesso em: 24 fev. 2019. Citado na página 15.

DOCKER. **Introduction to Docker Compose**. 2017. Disponível em: <https://runnable.com/docker/introduction-to-docker-compose>. Acesso em: 25 jan. 2019. Citado na página 19.

DOCKER. **Docker Engine Sparked the Containerization Movement**. 2018. Disponível em: <https://www.docker.com/products/docker-engine>. Acesso em: 6 fev. 2019. Citado na página 17.

FERREIRA, U. J. S. F. et al. Análise de tecnologias de virtualização e hardware de baixo custo para infraestutura de nuvem de pequeno porte. Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Norte, 2017. Citado na página 13.

FILHO, A. C. de A.; GAMA, K. Estudo comparativo entre docker swarm e kubernetes para orquestração de contêineres em arquiteturas de software com microsserviços. 2016. Citado na página 18.

GARTNET RESEARCH. **Leading The IoT: Gartner Insights on How to Lead in a Connected World**. 2017. Disponível em: <https://www.gartner.com/imagesrv/books/iot/iotEbook_digital.pdf>. Acesso em: 07 abr. 2019. Citado na página 8.

GERMANO, E.; GOIÁS, G. de; NARCISO, M. G. Encapsulando um sistema gerenciador de ensaios de genotipagem e de marcadores moleculares em containers docker. **Embrapa Informática Agropecuária-Comunicado Técnico (INFOTECA-E)**, Campinas: Embrapa Informática Agropecuária, 2018., 2018. Citado na página 17.

HENRIQUE, L. Ufrj - eel878 - redes de computadores i - primeiro semestre de 2018. 2018. Acessado: 03 oct. 2018. Disponível em: <https://www.gta.ufrj.br/ensino/eel878/redes1-2018-1/trabalhos-vf/sdn/>. Acesso em: 03 oct. 2018. Citado na página 22.

KREUTZ, D. et al. Software-defined networking: A comprehensive survey. **Proceedings of the IEEE**, Ieee, v. 103, n. 1, p. 14–76, 2015. Citado nas páginas 8, 21 e 23.

KWANG-MAN, K. et al. Efficient deployment of service function chains (sfcs) in a self-organizing sdn-nfv networking architecture to support iot. In: IEEE. **2018 Tenth International Conference on Ubiquitous and Future Networks (ICUFN)**. [S.l.], 2018. p. 650–653. Citado na página 8.

LAWAL, B. H.; NURAY, A. Real-time detection and mitigation of distributed denial of service (ddos) attacks in software defined networking (sdn). In: IEEE. **2018 26th Signal Processing and Communications Applications Conference (SIU)**. [S.l.], 2018. p. 1–4. Citado na página 21.

LEARN, W. Y. W. Linux containers: why they're in your future and what has to happen first. 2014. Citado nas páginas 13 e 14.

LIU, D.; ZHAO, L. The research and implementation of cloud computing platform based on docker. In: IEEE. **2014 11th International Computer Conference on Wavelet Actiev Media Technology and Information Processing (ICCWAMTIP)**. [S.l.], 2014. p. 475–478. Citado na página 17.

LUCIO, J. P. D. et al. Análise comparativa entre arquitetura monolítica e de microsserviços. Florianópolis, SC, 2017. Citado na página 17.

MONTEIRO, L.; XAVIER, L.; VALENTE, M. T. Uma caracterizaç ao em larga escala da arquitetura de sistemas docker. **networks**, v. 6, n. 5000, p. 5000, 2017. Citado na página 16.

MONTEIRO, L. de A.; ALMEIDA, W. H. C. Cluster de alta disponibilidade com docker swarm. 2017. Citado na página 17.

NAIK, N. Building a virtual system of systems using docker swarm in multiple clouds. In: IEEE. **2016 IEEE International Symposium on Systems Engineering (ISSE)**. [S.l.], 2016. p. 1–3. Citado na página 18.

OJO, M.; ADAMI, D.; GIORDANO, S. A sdn-iot architecture with nfv implementation. In: IEEE. **Globecom Workshops (GC Wkshps), 2016 IEEE**. [S.l.], 2016. p. 1–6. Citado nas páginas 11, 23 e 24.

OPSERVICES. **AFINAL, O QUE É DOCKER?** 2018. Disponível em: <https://www.opservices.com.br/o-que-e-docker/>. Acesso em: 01 oct. 2018. Citado na página 16.

RAZA, M. **Containers vs Virtual Machines: What's The Difference?** 2018. Citado na página 15.

ROTHENBERG, C. E. et al. Openflow e redes definidas por software: um novo paradigma de controle e inovação em redes de pacotes. **Cad. CPqD Tecnologia, Campinas**, v. 7, n. 1, p. 65–76, 2010. Citado na página 20.

SAKYA, R. **HOW SDN CAN HELP IOT - USING SOFTWARE DEFINED NETWORKS WITH IOT - EDGEWATER NETWORKS**. 2018. Disponível em: <https://www.edgewaternetworks.com/blog/how-sdn-can-help-iot-using-software-defined-networks-with-iot>. Acesso em: 24 sep. 2018. Citado na página 24.

SILVA, A. C. d. Virtualização: um estudo sobre conceitos e técnicas. Universidade Federal Fluiminense, 2017. Citado na página 13.

SILVA, L. J. Internet das coisas. **Engenharia Elétrica Telemática-Pedra Branca**, 2017. Citado na página 11.

SINGH, P. **Docker Containers and IoT Applications**. 2017. Disponível em: <https://iotbytes.wordpress.com/docker-containers-and-iot-applications/>. Acesso em: 18 jan. 2019. Citado na página 19.

THUBERT, P.; PALATTELLA, M. R.; ENGEL, T. 6tisch centralized scheduling: When sdn meet iot. In: **Proc. of IEEE Conf. on Standards for Communications & Networking (CSCN'15)**. [S.l.: s.n.], 2015. Citado na página 9.

TOMKIS, R. **how Software-Defined Networks (SDN) Works - Ciena**. 2018. Acessado: 25 oct. 2018. Disponível em: <https://www.ciena.com/insights/how/How-SDN-Works.html>. Acesso em: 25 oct. 2018. Citado na página 21.

TOZZI, C. **Why the Internet of Things Needs Docker Containers**. 2018. Disponível em: <https://containerjournal.com/2018/01/31/why-the-internet-of-things-needs-docker-containers/>. Acesso em: 16 jan. 2019. Citado na página 19.

VILALTA, R. et al. End-to-end sdn orchestration of iot services using an sdn/nfv-enabled edge node. In: IEEE. **2016 Optical Fiber Communications Conference and Exhibition (OFC)**. [S.l.], 2016. p. 1–3. Citado na página 8.

WAN, J. et al. Software-defined industrial internet of things in the context of industry 4.0. **IEEE Sensors Journal**, IEEE, v. 16, n. 20, p. 7373–7380, 2016. Citado na página 9.

YADAV, R.; SOUSA, E.; CALLOU, G. Performance comparison between virtual machines and docker containers. **IEEE Latin America Transactions**, IEEE, v. 16, n. 8, p. 2282–2288, 2018. Citado na página 14.

YASSEIN, M. B.; ABUEIN, Q.; ALASAL, S. A. Combining software-defined networking with internet of things: Survey on security and performance aspects. In: IEEE. **Engineering & MIS (ICEMIS), 2017 International Conference on**. [S.l.], 2017. p. 1–7. Citado na página 8.