Uma Arquitetura para Emulação Escalável em Névoa

Antonio Coutinho[1], Elisangela Carneiro[1], Fabíola Greve[2], Cássio Prazeres[2]

[1]Departamento de Tecnologia – Universidade Estadual de Feira de Santana (UEFS)
Feira de Santana, Bahia 44036–900

[2]Departamento de Ciência da Computação – Universidade Federal da Bahia
Salvador, Bahia 40170–110

{augusto,elisangela}@ecomp.uefs.br,{fabiola,cprazeres}@ufba.br

Abstract. *The deployment of IoT solutions employing a fog architecture requires a widely distributed compute infrastructure. However, the absence of a publicly available fog testbed makes the validation of fog solutions a challenging issue. Currently, fog prototyping environments are adapted from cloud middlewares or network simulators to the evaluation of solutions in limited scenes. This paper presents a scalable architecture to enable the deployment and testing of fog components in virtualized environments. Its design is compatible with real world technologies and meets the requirements of low cost, flexible setup and support third-party systems through standard interfaces. In addition, future developments and research directions are discussed.*

Resumo. *O desenvolvimento de aplicações IoT empregando uma arquitetura em névoa exige uma infraestrutura de computação amplamente distribuída. No entanto, a ausência de um testbed de névoa publicamente disponível torna a validação de soluções um problema desafiador. Atualmente, ambientes de prototipagem de névoa são adaptados de middlewares de nuvem ou simuladores de rede para a avaliação de soluções em cenários limitados. Este artigo apresenta uma arquitetura escalável para permitir o desenvolvimento e teste de componentes de névoa em ambientes virtualizados. O projeto é compatível com tecnologias do mundo real e atende aos requisitos de configuração flexível, baixo custo e suporte a sistemas de terceiros através do uso de interfaces padronizadas. Além disso, são discutidos trabalhos futuros e as direções de pesquisa.*

1. Introdução

Um modelo centralizado baseado em nuvem tornou-se a abordagem padrão para a internet das coisas (*Internet of Things*, IoT). Atualmente, tem surgido diferentes iniciativas que apoiam uma mudança do modelo centralizado para uma arquitetura descentralizada com a finalidade de superar as limitações das plataformas IoT baseadas em nuvem [Coutinho et al. 2016]. As propostas apresentadas em [Bonomi et al. 2012] [Dinh et al. 2013] [Hu et al. 2015] [Satyanarayanan 2017] podem ser resumidas através de suas estratégias convergentes: uso de pequenas nuvens localizadas com suporte a virtualização, distribuição de recursos computacionais em larga escala, fornecimento de capacidades IoT baseado em modelos de serviços em nuvem e na oferta de serviços exclusivos usando informações que estão presentes na rede local ou em suas proximidades.

O conceito de computação em névoa apresentado em [Bonomi et al. 2012] é definido como uma plataforma virtualizada que oferece serviços entre dispositivos finais e centros de dados da computação em nuvem. Também oferece suporte ao conceito de computação de borda [Satyanarayanan 2017] de forma escalável e integrada com dispositivos de rede, como comutadores, roteadores e gateways IoT.

A abordagem distribuída de computação em névoa reduz a quantidade de dados transferidos para o núcleo da rede, capturando e processando os dados necessários localmente em cada dispositivo de borda. O suporte em baixa latência torna possível a análise em tempo real dos dados localizados na borda, distribuindo o processo analítico através da rede [Bonomi et al. 2014].

A arquitetura do sistema de névoa é distribuída hierarquicamente em uma infraestrutura de rede de *n-tier* com no mínimo três camadas. A Figura 1 apresenta uma visão do sistema de névoa com três camadas: a camada da nuvem, a camada da névoa e a camada da IoT/usuários finais.

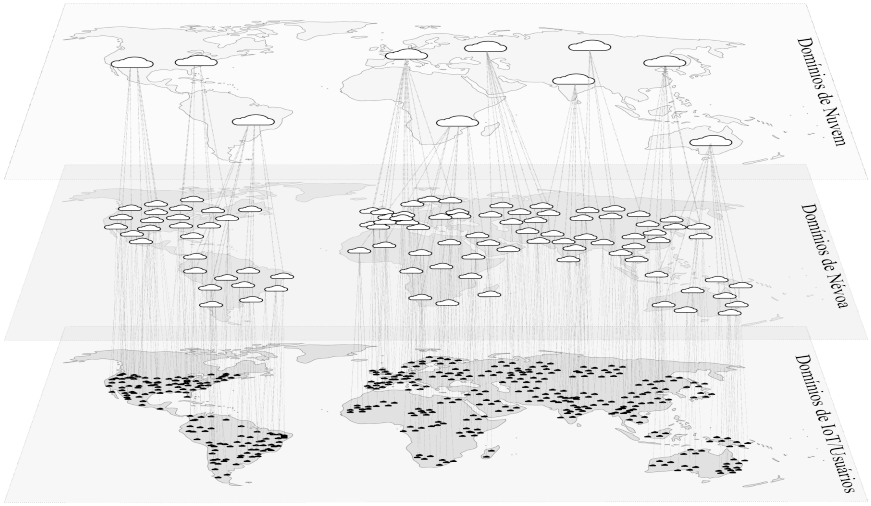

Figura 1. Arquitetura horizontal do sistema de névoa.

Localizados na borda da rede, os dispositivos sensores e atuadores instalados em veículos, parques, indústrias, prédios e ruas podem fornecer valiosos fluxos de dados para aplicações de monitoramento, controle, análise e previsão que funcionam em nós de gateway próximos à borda. Sua localização depende de fatores como economia de energia, tempo de resposta ou aplicativos com requisitos de baixa latência. Em algumas aplicações, os dispositivos IoT inteligentes embarcados podem receber comandos de atuação ou enviar dados de sensoriamento sem a necessidade de gateways dedicados. Além disso, eles são capazes de atuar como gateways de borda para outros dispositivos IoT ou usuários finais.

A camada da névoa é formada por um ou mais domínios de névoa, controlados pelos mesmos ou diferentes provedores. Na hierarquia do sistema, a camada da névoa torna os dados transmitidos acionáveis, significativos e valiosos através da filtragem de dados em diferentes níveis da infraestrutura. Esta arquitetura envolve a execução de serviços e aplicações baseadas em componentes distribuídos nos nós de névoa entre os dispositivos IoT e os centros de dados da nuvem.

Os serviços que gerenciam e suportam recursos virtualmente ilimitados para aplicativos IoT são executados na camada da nuvem. Essas aplicações podem usar recursos tanto da nuvem quanto da névoa para fornecer soluções inteligentes e de ponta para usuários finais. Isso pode ser alcançado por soluções eficientes que distribuem demandas de execução de tarefas em nós de névoa ótimos, bem localizados e disponíveis. O objetivo principal é fornecer níveis toleráveis de latência, ao mesmo tempo em que otimiza o desempenho das aplicações da névoa.

Espera-se que os ambientes de névoa suportem milhões de dispositivos IoT e de usuários finais. Eles podem envolver um grande número de aplicativos, domínios e nós de névoa. Além disso, várias aplicações podem ter uma grande quantidade de componentes. Assim, os sistemas de névoa precisam ser operacionais em larga escala e escaláveis de forma elástica, ou seja, capazes de diminuir e aumentar quando necessário.

Um problema atual nesta área é a falta de ambientes de suporte para prototipagem e teste de serviços, componentes e aplicações em larga escala [Mouradian et al. 2017] [Coutinho et al. 2018]. Até onde sabemos, não há ambientes de testes para névoa disponíveis que possam auxiliar os pesquisadores a projetar e verificar algoritmos distribuídos em uma escala IoT real. Neste momento, simuladores de rede e *middleware* de nuvem são adaptados para permitir uma avaliação experimental de soluções de névoa em ambientes limitados, cenários específicos e condições restritivas. No entanto, testar e validar uma arquitetura com poucos dispositivos e nós não garante que ela seja propriamente executada sobre um ambiente em larga escala, no que diz respeito à qualidade e desempenho da solução desenvolvida.

Em trabalho anterior [Coutinho et al. 2018], foi proposto o *Fogbed*, que é um sistema de emulação baseado em *software* aberto para prototipagem de componentes de névoa em ambientes virtualizados. Suas funcionalidades permitem a prototipagem e a implementação por desenvolvedores de soluções de névoa em um ambiente *desktop*.

Este artigo apresenta uma arquitetura que estende o *Fogbed* para permitir o desenvolvimento e o teste de componentes de névoa de forma escalável. Isso é possível pela criação, configuração e conexão de instâncias virtuais distribuídas em diferentes máquinas em ambiente de rede e executando contêineres do *Fogbed*.

O restante do artigo está organizado como segue. A Seção 2 apresenta trabalhos relacionados. A Seção 3 aborda os requisitos e as escolhas tecnológicas empregadas na solução. A Seção 4 ilustra a arquitetura da emulação distribuída proposta. As conclusões e trabalhos futuros são apresentados na Seção 5.

2. Trabalhos Relacionados

Recentemente, trabalhos fundamentais como [Bonomi et al. 2014], [Yi et al. 2015], [Dastjerdi and Buyya 2016], [OpenFog 2017b] tem proposto uma arquitetura de referência, requisitos de projetos e componentes de uma plataforma de névoa. [Byers 2017] discute importantes requisitos arquiteturais e técnicas para implementar redes IoT com suporte a névoa no contexto de exemplos de casos de uso. Em [Mouradian et al. 2017] é apresentada uma classificação estruturada das arquiteturas atuais, algoritmos, questões pendentes e direções de pesquisa para sistemas de névoa.

Os trabalhos existentes tem focado no projeto de uma adequada arquitetura de

computação em névoa, que pode servir como um modelo de desenvolvimento para diferentes plataformas de larga escala. Em novembro de 2015, o consórcio OpenFog [OpenFog 2017a] reuniu empresas líderes de TI e universidades para desenvolver uma arquitetura padrão aberta para computação em névoa. A primeira versão da arquitetura de referência OpenFog [OpenFog 2017b] foi lançada em fevereiro de 2017.

No entanto, para validar uma arquitetura e ou aplicações baseadas em névoa é necessário implementar seus componentes em um ambiente adequado e que permita avaliar aspectos tais como segurança, desempenho e escalabilidade.

A Cisco foi uma das primeiras empresas a fornecer um ambiente de teste aberto para aplicações de névoa através de laboratórios virtuais chamados IOx Sandboxes [Cisco System Inc. 2017b]. Os principais componentes do ambiente de aplicação Cisco IOx são: (i) o Cisco IOx [Cisco Systems Inc. 2017], um sistema operacional de rede que torna os recursos de processamento e armazenamento disponíveis para uma grande variedade de aplicativos IoT hospedados em máquinas virtuais (VMs); (ii) o *Fog Director* [Cisco System Inc. 2017a], que fornece serviços centralizados e APIs RESTful para gerenciar, administrar, integrar e monitorar aplicativos e dispositivos Cisco IOx remotamente através da rede usando uma abordagem *desktop*; (iii) SDK e ferramentas de desenvolvimento, formados por um conjunto de pacotes de *software* usados por desenvolvedores de terceiros para construir e integrar aplicações. O Cisco IOx *Application Framework* (CAF) também suporta sistemas de código aberto e aplicações baseadas em instâncias Linux executando sobre um *hypervisor*.

O Cisco IOx Sandbox permite o desenvolvimento e teste de aplicações de névoa usando um desktop ou *notebook* com acesso à internet. Um desenvolvedor pode reservar e acessar remotamente o ambiente simulado para instalar, desenvolver e executar aplicativos IOx. O ambiente IOx Lab (versão 1.5) consiste em dois nós IOx-CAF baseados na VM Linux Yocto 1.8 e um nó IOx-*Fog Director* baseado na VM Linux Ubuntu 14.04.

A plataforma IOx executada em ambiente *sandbox* é uma ferramenta adequada para promover o ensino e a aprendizagem de tecnologias IoT proprietárias da Cisco. No entanto, o teste de soluções com poucos dispositivos e nós não garante que eles funcionam corretamente em um ambiente de larga escala, em termos de qualidade e desempenho da solução entregue. A este respeito, é necessária uma infraestrutura de computação amplamente distribuída que reúne recursos de rede, computação e armazenamento em uma hierarquia de níveis entre a fonte de dados e a nuvem.

Na ausência de *testbeds* de larga escala para névoa, uma alternativa é adaptar os atuais *testbeds* para IoT de modo a suportar avaliação experimental de soluções para névoa. Os maiores *testbeds* de IoT, tais como [Adjih et al. 2015] e [Sanchez et al. 2014], são providos com milhares de dispositivos IoT distribuídos geograficamente. Os usuários autorizados podem executar aplicações sobre um grande número de recursos para avaliar protocolos de baixo nível, técnicas e serviços em larga escala. No entanto, uma vez que esses ambientes não visam aplicações de névoa, é necessário um maior esforço para configurar e implementar experimentos de névoa.

Apesar da relevância do critério de escalabilidade, os trabalhos encontrados na literatura envolvendo soluções de névoa e tecnologias reais foram avaliados em cenários de pequena escala. Outros trabalhos que empregam a computação em névoa em cenários

maiores foram avaliados através de simulações numéricas como [Fricker et al. 2016], [Sarkar and Misra 2016], [Liu et al. 2017].

Embora importante, uma simulação em larga escala não trata os problemas relacionados à sua implementação prática. Além da escalabilidade, os algoritmos de névoa precisam operar em condições e parâmetros realistas do sistema. Por exemplo, um algoritmo de agendamento de tarefas em redes veiculares precisa ser operacional para o volume e padrões de tráfego esperado em cidades inteligentes. Existem iniciativas e tecnologias de código aberto que podem ser usadas para criar *testbeds* para névoa baseados no mundo real. Por exemplo, Soft-IoT [Prazeres and Serrano 2016] é um *framework* que suporta a computação em névoa e borda. Em [Villari et al. 2017], o desenvolvimento e a orquestração de microserviços distribuídos na borda usa um protótipo de plataforma de névoa constituída sobre quatro nuvens federadas e implementadas com o *OpenStack*.

No entanto, os sistemas de névoa reais tem sido validados em pequena escala, sem garantias de que eles funcionarão bem em larga escala, deixando este aspecto como um desafio a ser tratado futuramente. Isso continua sendo uma abordagem padrão, tendo em vista os desafios envolvidos na execução e operação de avaliações experimentais de névoa em larga escala. Embora desejável, o uso de um *testbed* é custoso, demorado e, em alguns casos, difícil de acessar, configurar e operar.

No contexto atual, alternativas econômicas de baixo custo devem ser consideradas. As simulações em larga escala continuam sendo uma opção acessível que permite a comparação de diferentes algoritmos, além de eliminar políticas e estratégias ineficazes. Para este fim, a adaptação de *frameworks* ou simuladores de código aberto existentes é uma abordagem comum na área de pesquisa em névoa.

Em [Brennand et al. 2016] e [Brennand et al. 2017] são apresentados mecanismos baseados em névoa para sistemas de transporte inteligentes (*Intelligent Transport Systems*, ITS) que visam gerenciar o congestionamento de tráfego em redes ad hoc veiculares (*Vehicular Ad Hoc Networks*, VANETs). Ambos ambientes de simulação foram desenvolvidos através do OMNET ++ que não é um simulador de rede propriamente dito, mas um *framework* e bibliotecas extensíveis para a construção de simuladores de rede.

Simuladores específicos para avaliar *frameworks* de computação em névoa foram projetados em [Hong et al. 2016], [Etemad et al. 2017], [Sonmez et al. 2017]. Em [Dastjerdi and Buyya 2016], um estudo de caso sobre o gerenciamento inteligente de tráfego estendeu o CloudSim [Calheiros et al. 2011] com capacidades de névoa para avaliar o desempenho das aplicações em termos de consumo de largura de banda e tempo de resposta. Este trabalho foi a base do iFogSim [Gupta et al. 2016], um simulador que suporta recursos de névoa e borda em diferentes cenários. Ele permite a avaliação de políticas de gerenciamento de recursos, analisando sua influência sobre o consumo de energia, os custos operacionais, a latência e o congestionamento de rede. Em [Lopes et al. 2017] MyiFogSim estendeu o iFogSim para suporte a mobilidade através da migração de VMs entre *cloudlets*. No entanto, os ambientes específicos de névoa citados não apresentam suporte a simulações em larga escala.

Em resumo, as soluções de névoa continuam a ser desenvolvidas como prova de conceito, implementadas em ambientes limitados, cenários específicos e condições restritas. Pesquisas atuais relacionadas a disponibilização de um *testbed* escalável com suporte

a tecnologias reais e computação em névoa podem ajudar a validar componentes, algoritmos e aplicações. Essas arquiteturas pré-testadas e validadas ajudarão os provedores de serviços a atingir seus objetivos, minimizando os riscos envolvidos na mudança para o modelo de computação em névoa.

3. Requisitos e Soluções Tecnológicas

A presente arquitetura é baseada em soluções que atendem aos seguintes requisitos: (1) baixo custo; (2) configuração rápida e flexível; (3) suporte para realizar protocolos e serviços do mundo real; e (4) execução escalável. No restante desta seção, são descritas soluções de código aberto que permitem a implantação dos principais componentes da névoa de acordo com esses requisitos. Eles permitem o teste de tecnologias reais em um ambiente repetível e controlável. Uma arquitetura que emprega essas soluções para criar ambientes de *testbed* de forma distribuída e escalável é apresentada na Seção 4.

3.1. Plataforma de Conteiner *Docker*

O Docker [Docker 2017] é uma plataforma de *software* que usa uma tecnologia baseada em *kernel* Linux chamada conteinerização. Um contêiner é um pacote independente e executável que inclui o que é necessário para executar um aplicativo, como código executável, ambientes de tempo de execução, configurações, ferramentas e bibliotecas do sistema. As aplicações em execução dentro de um contêiner *Docker* utilizando virtualização no nível do sistema operacional são semelhantes às instâncias de VM tradicionais. No entanto, os contêineres não encapsulam um sistema operacional completo. A *engine Docker* compartilha o *kernel* do *host* e isola os recursos necessários à execução da aplicação usando uma abordagem leve baseada em *cgroups* do Linux e *namespace* do *kernel*.

O uso de contâineres tem se popularizado entre os desenvolvedores pela sua flexibilidade, portabilidade e escalabilidade. Mesmo as aplicações complexas podem ser conteinerizadas e transmitidas na internet em um arquivo de tamanho menor em comparação com as imagens de VM. Assim, é possível criar uma imagem de contêiner localmente, armazená-la na nuvem ou distribuir e executar automaticamente réplicas na rede.

O sistema nativo de orquestração do *Docker* [Docker built-in orchestration 2017] permite implementar e executar imagens de contêiner em máquinas distribuídas. Este esquema possibilita uma plataforma distribuída, independente da infraestrutura, para execução de aplicações conteinerizadas em escala. A API *Docker* pode ser usada para implementação e gerenciamento de aplicações em um grupo de *self-healing engines* chamado de *swarm* [Docker built-in orchestration 2017]. Assim, da mesma forma que executa um único contêiner, o ambiente *Docker* pode iniciar um processo replicado, distribuído e balanceado em um *swarm* de *Docker engines*.

A Figura 2 ilustra o modo *swarm* do *Docker*, onde as setas definem a hierarquia de comunicação e controle entre os componentes. O primeiro nó que se une ao *swarm* assume a função de gerenciamento, o que significa que ele aceita comandos e escalonamento de tarefas. À medida que mais nós se juntam ao *swarm*, eles se tornam nós trabalhadores por padrão. Isso implica que eles executam os contêineres despachados pelo gerente. Na Figura 2, cada tipo de contêiner (A ou B) executa um serviço diferente como, por exemplo, um serviço de cadastro e um banco de dados. Cada contêiner ou serviço pode ser replicado em diferentes nós trabalhadores, dispostos abaixo de cada

contêiner na Figura 2. Além disso, para o balanceamento de carga eficiente dos serviços, é possível definir nós gerenciadores adicionais que irão participar em um grupo consenso *Raft* [Ongaro and Ousterhout 2014]. Os nós gerentes usam um armazenamento *Raft* otimizado, no qual as leituras são realizadas diretamente da memória principal para tornar o desempenho do escalonamento mais rápido. Os nós do *swarm* trocam informações da rede de *Overlay* usando um protocolo *gossip*.

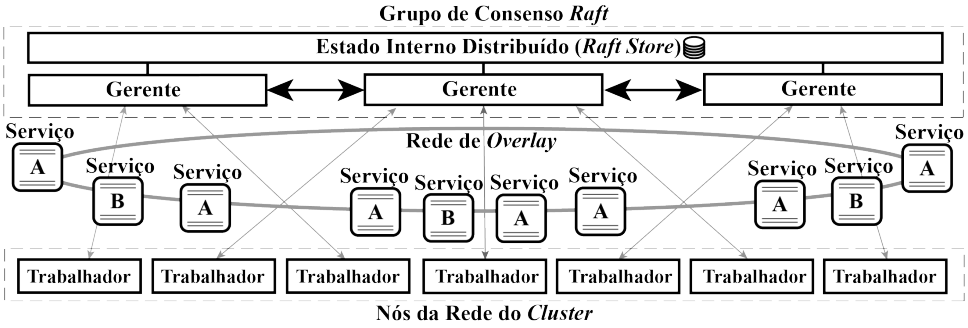

Figura 2. Uma visão geral do modo *swarm* do *Docker*.

3.2. Emulador de Rede *Fogbed*

O *Fogbed* [Coutinho et al. 2018] é um projeto de *software* que estende o *framework Mininet* para criar *testbeds* para névoa em ambientes virtualizados. Usando uma abordagem *desktop*, o *Fogbed* permite a implementação de nós de névoa como contêineres *Docker* sob diferentes configurações de rede. A API do *Fogbed* fornece funcionalidades para adicionar, conectar e remover contêineres dinamicamente da topologia da rede. Essas características permitem a emulação da infraestrutura de nuvem e névoa, na qual é possível iniciar e interromper instâncias de computação em qualquer momento. Além disso, é possível alterar as limitações de recursos em tempo de execução para um contêiner, como o tempo da CPU e memória disponível.

O ambiente de emulação *Fogbed* pode ser criado pela implementação de nós virtuais, comutadores virtuais, conexões virtuais e instâncias virtuais em um ambiente de rede virtual em execução em uma máquina *host*. A configuração flexível é alcançada através do uso de imagens de contêiner *Docker* pré-configuradas. Cada imagem de contêiner compreende parte de uma aplicação distribuída, bem como seus serviços e protocolos. Diferentes tipos de imagens de contêiner podem ser usados para instanciar nós virtuais. Um exemplo de uma emulação de ambiente de névoa é mostrado na Figura 3, onde três tipos de imagens de contêiner foram usadas para instanciar diferentes nós virtuais. O ambiente descrito na Figura 3 é posteriormente explicado na Seção 4.

A instância virtual é uma abstração que permite o gerenciamento de um conjunto de nós virtuais e comutadores virtuais relacionados como uma única entidade. Uma aplicação de névoa e seus serviços são executados em um ou mais nós virtuais dentro de uma instância virtual. Dependendo do tipo contêiner baseado, eles podem ser uma instância de nuvem virtual (*Virtual Cloud Instance*, VCI), uma instância de névoa virtual (*Virtual Fog Instance*, VFI) ou uma instância de borda virtual (*Virtual Edge Instance*, VEI). O processo do sistema de gerenciamento é responsável pela inicialização das instâncias no ambiente de emulação *Fogbed*. Ele primeiro executa um script que define a

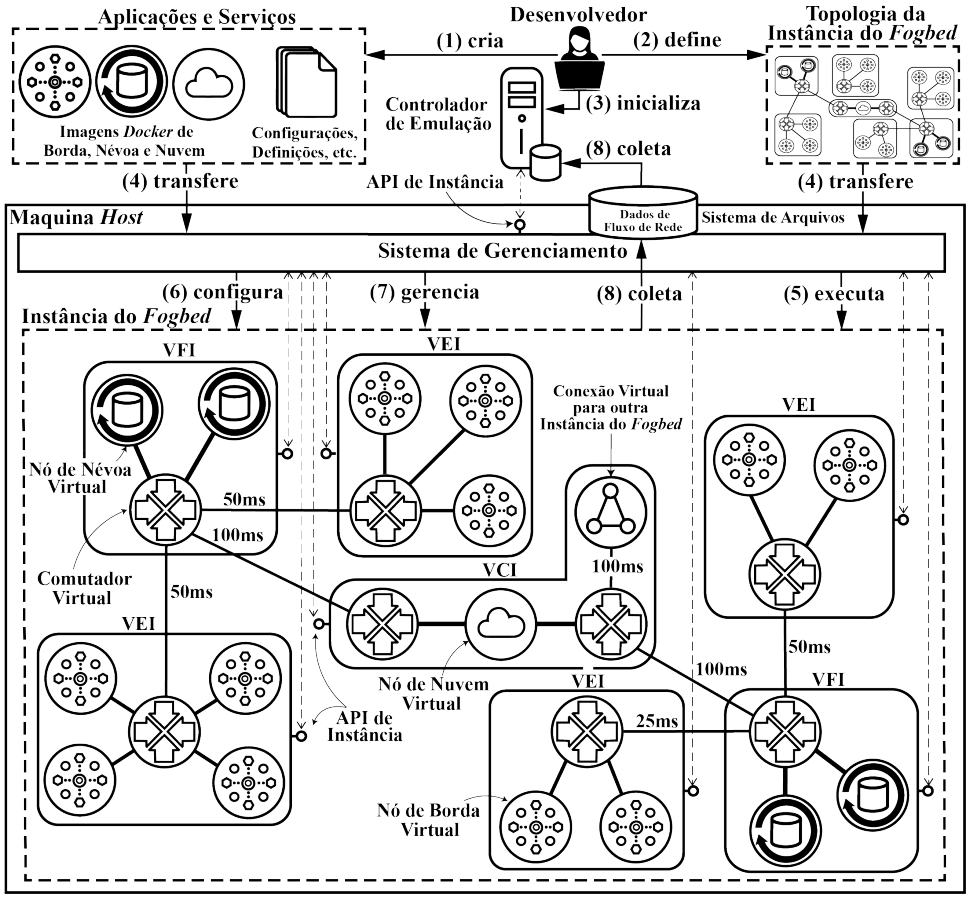

Figura 3. Fluxo de Trabalho das instâncias de emulação no *Fogbed*.

topologia de rede entre instâncias virtuais. A comunicação entre uma instância virtual e o sistema de gerenciamento é realizada através de uma API de instância bem definida.

4. Arquitetura de Emulação Distribuída

O objetivo deste trabalho é projetar uma arquitetura que permita a implementação dos principais componentes de um ambiente de névoa de forma escalável. A Figura 4 mostra a arquitetura de emulação distribuída e os componentes do sistema. A discussão nesta seção tem como foco descrever os componentes principais para permitir uma emulação escalável utilizando o *Fogbed*.

A escalabilidade foi alcançada através do particionamento do ambiente de névoa emulado através de instâncias do *Fogbed* executadas em máquinas *hosts* distribuídas. Como qualquer aplicação baseada em Linux, o *Fogbed* pode ser conteinerizado como uma imagem *Docker*. Essas imagens do *Fogbed* são replicadas, distribuídas e instanciadas em um conjunto de máquinas empregando o modo *swarm* do *Docker*. A API do *Docker engine* é usada para criar um *swarm* escalável de instâncias do *Fogbed* executadas em vários nós para implementar e testar componentes de névoa.

O *swarm* é criado a partir de uma única imagem do *Fogbed*. Ao iniciar, cada máquina *host* executa o serviço de gerenciamento local que espera o que é necessário para executar sua parte da emulação como imagens de contêiner, *script* de topologia e

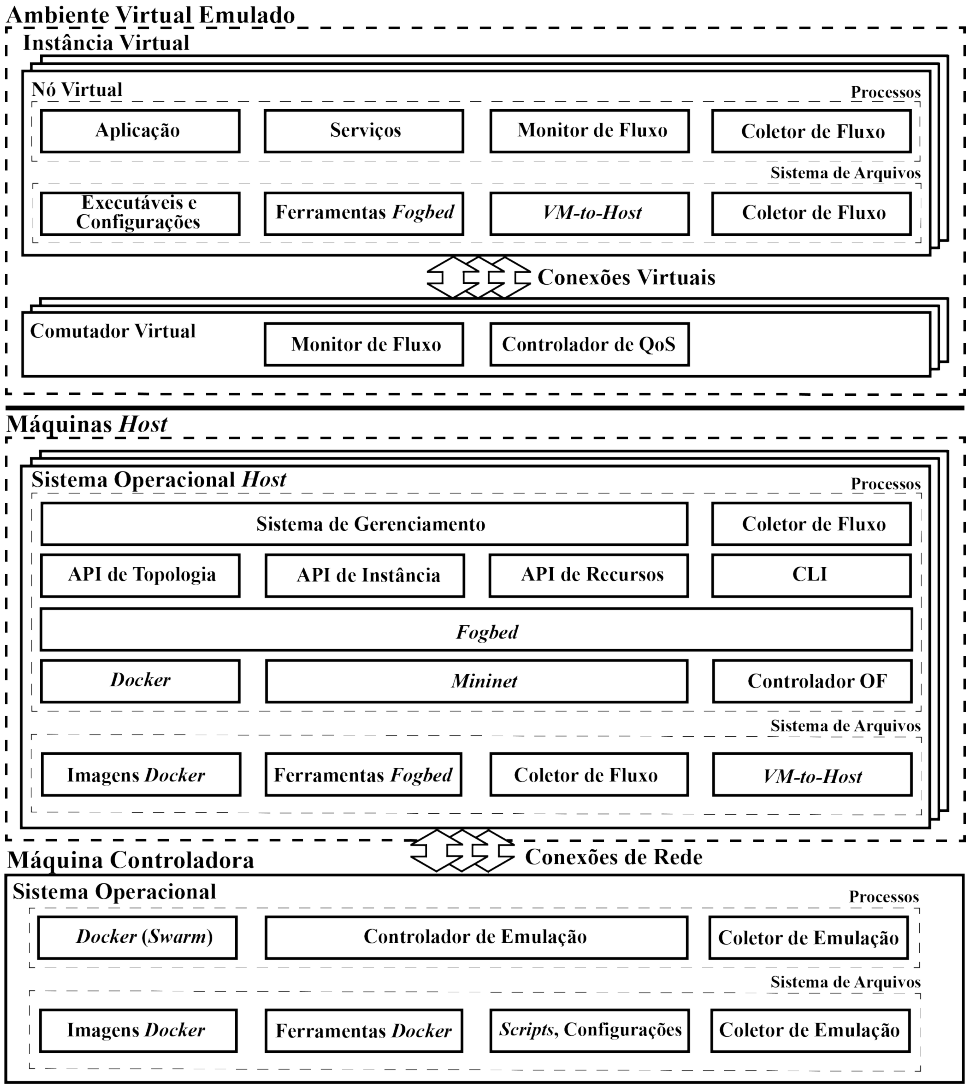

Figura 4. Uma visão geral da arquitetura de emulação distribuída.

configurações. Após este estágio, um nó do Fogbed está pronto para ser configurado remotamente e vinculado com outras instâncias, dispositivos ou redes de nuvem remotas através da configuração de conexões *proxy* virtuais.

Dentro de cada instância do *Fogbed*, os nós virtuais e sua infraestrutura de rede são emulados. O processo controlador executado na máquina controladora da emulação usa a API de instância remota do *Fogbed* para iniciar e controlar remotamente a execução de diferentes instâncias virtuais distribuídas na rede local.

A Lista 1 mostra um exemplo de *script* para definir instâncias de nó do *Fogbed*. Ele usa a API de topologia do *Fogbed* para definir instâncias virtuais e suas conexões com diferentes propriedades de *link* (Lista 1, linhas 1-14). A API de recurso permite que cada instância virtual use um modelo de recurso específico (Lista 1, linhas 15-20). O exemplo adiciona uma API de instância específica a cada tipo de instância virtual (Lista 1, linhas 21-35). Além disso, os desenvolvedores podem implementar suas próprias interfaces de gerenciamento em cima de uma API de instância virtual.

A Figura 3 e a Figura 5 representam o fluxo de trabalho em alto nível de um de-

```
fb1 = net.addFogbedInstance("Domain Fog")
fb2 = net.addFogbedInstance("Domain Cloud")
e1 = net.addEdgeInstance("Edge gateway")
fb1.addInstance(e1)
f1 = net.addFogInstance("Fog device")
fb1.addInstance(f1)
c1 = net.addCloudInstance("Cloud gateway")
fb2.addInstance(c1)
s1 = net.addSwitch("switch from Edge to Fog")
s2 = net.addSwitch("switch from Fog to Cloud")
net.addLink(e1, s1, delay="10ms", loss=2)
net.addLink(f1, s1, delay="50ms")
net.addLink(f1, s2, delay="50ms")
net.addLink(c1, s2, delay="100ms")
r1 = ResModelA(max_cu=20, max_mu=40, max_su=60)
r1.assignInstance(e1)
r2 = ResModelB(max_cu=200, max_mu=150, max_su=200)
r2.assignInstance(f1)
r3 = ResModelC(max_cu=1000, max_mu=1500, max_su=2000)
r3.assignInstance(c1)
api1 = EdgeApi(port=8001)
api1.connectInstance(e1)
api1.start()
api2 = FogApi(port=8002)
api2.connectInstance(f1)
api2.start()
api3 = CloudApi(port=8003)
api3.connectInstance(c1)
api3.start()
api4 = FogRemoteApi(port=8004)
api4.connectInstance(fb1)
api4.start()
api5 = CloudRemoteApi(port=8005)
api5.connectInstance(fb2)
api5.start()
net.start()
```

Lista 1. Um exemplo de um *script* de topologia do Fogbed.

senvolvedor usando o ambiente. A Figura 5 mostra o ambiente de emulação distribuído enquanto a Figura 3 se concentra nos detalhes para iniciar uma única instância do *Fogbed*. Primeiro, o desenvolvedor fornece as imagens do contênier que serão usadas para configurar a emulação (1). Cada imagem de contênier inclui tudo o que é necessário para executar uma aplicação. Usando a API de topologia, o desenvolvedor define o ambiente distribuído no qual ele deseja testar a aplicação (2) e inicia o processo controlador de emulação com a definição de topologia (3). O processo controlador usa a API *Docker Swarm* para criar as instâncias do *Fogbed* necessárias, conforme definido no *script* de topologia da emulação. Após o *swarm Fogbed* ser iniciado, o processo controlador de emulação invoca o serviço de gerenciamento em cada instância do *Fogbed* para fazer o *upload* das imagens de contêiner, configurações e *script* da topologia (4). O sistema de gerenciamento local executa sua parte da topologia e conecta seu ambiente emulado usando a API de instância fornecida (5). A aplicação é iniciada em cada instância do *Fogbed* através do sistema de gerenciamento local, que executa os processos e os serviços necessários em cada nó virtual local (6). Assim, a aplicação pode ser gerenciada a partir da máquina controladora da emulação usando uma CLI remota para acessar cada instância do *Fogbed* (7). Além disso, as instâncias do sistema de gerenciamento podem se comunicar para executar políticas de gerenciamento. As estatísticas de fluxo da rede podem ser coletadas em cada instância do *Fogbed* e armazenadas para análise futura (8). Além disso, o desenvolvedor na máquina de controle da emulação pode acessar dados arbitrários de

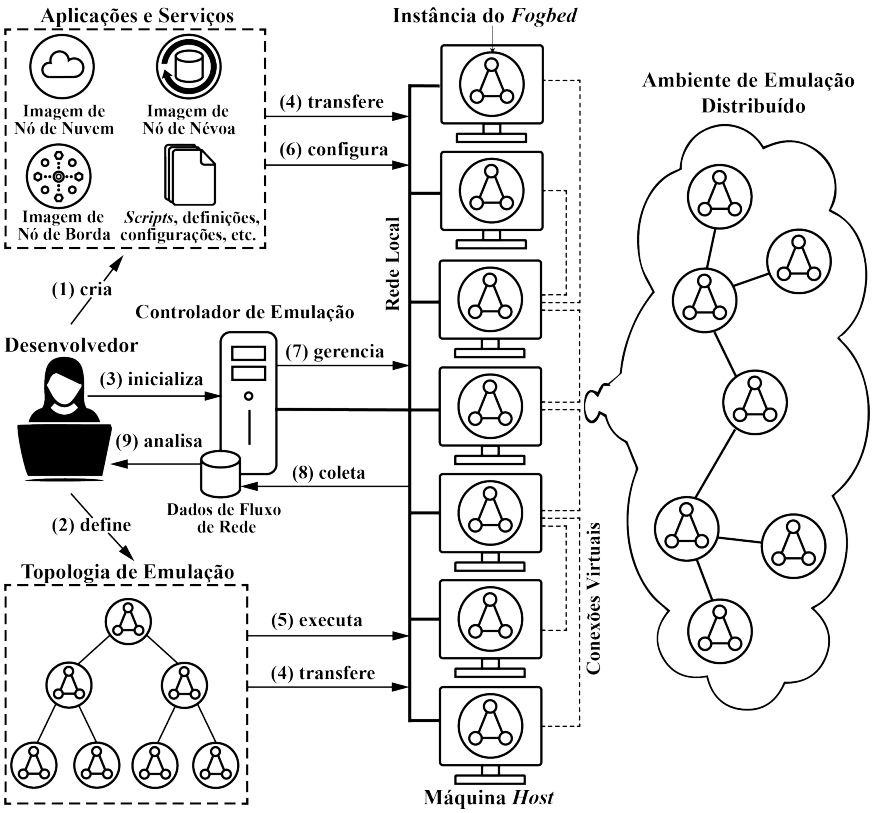

Figura 5. Fluxo de Trabalho da emulação distribuída no *Fogbed*.

monitoramento gerados pela plataforma (9).

Dentro de uma instância do *Fogbed*, o monitoramento do tráfego da rede pode ser implementado executando monitores de fluxo para cada interface de rede em nós virtuais e comutadores virtuais. Os dados de tempo de execução desses processos são salvos na pasta *VM-to-Host* que está vinculada à mesma pasta em sua máquina *host*.

O *daemon* de captura executado em um nó virtual é responsável por coletar e salvar estatísticas de fluxo dos monitores de fluxo do nó virtual. É necessário iniciar um monitor de fluxo para cada interface de rede. O tráfego nos nós virtuais pode se sobrepor ao tráfego monitorado em interfaces de comutadores virtuais.

No *Fogbed*, os comutadores virtuais não são contêiners *Docker*. Assim como no *Mininet*, eles são criados na máquina *host*. Consequentemente, os comandos do monitor de fluxo e *daemons* de captura devem ser executados na máquina *host*. Um *daemon* de captura de fluxo em execução em uma máquina *host* local é responsável por coletar e salvar estatísticas de fluxo de um comutador virtual local. Cada *daemon* de captura de fluxo guarda os registros de fluxo em um diretório diferente. Desta forma, é possível separar os dados coletados de diferentes comutadores e nós virtuais.

5. Conclusões e Trabalhos Futuros

Este artigo propôs uma arquitetura de emulação em névoa distribuída baseada em tecnologias abertas amplamente utilizadas pela comunidade de TI e desenvolvidas para uso em sistemas escaláveis. A arquitetura foi projetada para prototipagem e testes de componen-

tes de névoa executados em ambientes virtualizados como um *cluster* de máquinas em rede local ou sistemas em nuvem. Uma discussão mais geral sobre o ambiente *Fogbed*, pode ser encontrada em [Coutinho et al. 2018]. Os problemas levantados abaixo estão relacionados com a implementação de uma emulação escalável usando o *Fogbed*.

A conteinerização de um ambiente emulado do *Fogbed* e sua execução em um *cluster* de máquinas empregando o modo *swarm* do *Docker* torna a arquitetura proposta escalável. Por exemplo, quando a capacidade de memória de uma máquina *host* é preenchida por instâncias virtuais, um novo nó trabalhador ou máquina *host* pode ser adicionada na rede para suportar novos nós virtuais. No entanto, o controlador de emulação centralizado limita o desempenho da emulação. Em simulações com muitas instâncias do *Fogbed*, o tempo para iniciar o ambiente pode aumentar para um nível impraticável. A escalabilidade da arquitetura pode ser melhorada distribuindo e balanceando as funcionalidades do controlador de emulação entre diferentes máquinas *host* no ambiente emulado.

O uso das mesmas imagens de contêiner pré-configuradas para iniciar mais de uma instância facilita o teste de soluções que lidam com fluxos de dados massivos tais como aplicações em *big data*. Através de um procedimento simples é possível criar ambientes de névoa com várias fontes de dados IoT a partir de poucas imagens de contêiner. Essas funções podem ser controladas por um sistema distribuído de gerenciamento e orquestração com diferentes requisitos para instâncias de nuvem, névoa e borda.

Aspectos como segurança, tolerância à falhas e gerenciamento confiável podem ser investigados através do desenvolvimento de funcionalidades para permitir a inserção de ataques simulados e problemas no ambiente emulado. Finalmente, pesquisas adicionais sobre a arquitetura e seus componentes podem permitir a execução de experimentos de névoa realistas e reproduzíveis.

Referências

Adjih, C., Baccelli, E., Fleury, E., Harter, G., Mitton, N., Noel, T., Pissard-Gibollet, R., Saint-Marcel, F., Schreiner, G., Vandaele, J., et al. (2015). Fit iot-lab: A large scale open experimental iot testbed. In *Internet of Things (WF-IoT), 2015 IEEE World Forum on*, pages 459–464. IEEE.

Bonomi, F., Milito, R., Natarajan, P., and Zhu, J. (2014). Fog computing: A platform for internet of things and analytics. In *Big Data and Internet of Things: A Roadmap for Smart Environments*, pages 169–186. Springer.

Bonomi, F., Milito, R., Zhu, J., and Addepalli, S. (2012). Fog computing and its role in the internet of things. In *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, pages 13–16. ACM.

Brennand, C. A., Boukerche, A., Meneguette, R., and Villas, L. A. (2017). A novel urban traffic management mechanism based on fog. In *Computers and Communications (ISCC), 2017 IEEE Symposium on*, pages 377–382. IEEE.

Brennand, C. A., da Cunha, F. D., Maia, G., Cerqueira, E., Loureiro, A. A., and Villas, L. A. (2016). Fox: A traffic management system of computer-based vehicles fog. In *Computers and Communication (ISCC), 2016 IEEE Symposium on*, pages 982–987. IEEE.

Byers, C. C. (2017). Architectural imperatives for fog computing: Use cases, requirements, and architectural techniques for fog-enabled iot networks. *IEEE Communications Magazine*, 55(8):14–20.

Calheiros, R. N., Ranjan, R., Beloglazov, A., De Rose, C. A., and Buyya, R. (2011). Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and experience*, 41(1):23–50.

Cisco System Inc. (2017a). Fog Director. Disponível em: http://www.cisco.com/c/en/us/products/cloud-systems-management/fog-director/. Acesso em: 10 fev. 2018.

Cisco System Inc. (2017b). IOx Sandbox. Disponível em: https://developer.cisco.com/docs/sandbox/#iot. Acesso em: 10 fev. 2018.

Cisco Systems Inc. (2017). IOx Documentation. Disponível em: https://developer.cisco.com/docs/iox/. Acesso em: 10 fev. 2018.

Coutinho, A., Greve, F., Prazeres, C., and Cardoso, J. (2018). Fogbed: A rapid-prototyping emulation environment for fog computing. In *Communications Workshops (ICC Workshops), 2018 IEEE International Conference on*. IEEE.

Coutinho, A. A. T. R., Greve, F. G. P., and Carneiro, E. O. (2016). Computação em névoa: conceitos, aplicações e desafios. In *Minicursos - XXXIV Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*, pages 266–315. SBC.

Dastjerdi, A. V. and Buyya, R. (2016). Fog computing: Helping the internet of things realize its potential. *Computer*, 49(8):112–116.

Dinh, H. T., Lee, C., Niyato, D., and Wang, P. (2013). A survey of mobile cloud computing: architecture, applications, and approaches. *Wireless communications and mobile computing*, 13(18):1587–1611.

Docker (2017). project home page. Disponível em: https://www.docker.com. Acesso em: 20 set. 2017.

Docker built-in orchestration (2017). Swarm engine. Disponível em: https://docs.docker.com/engine/swarm/. Acesso em: 10 fev. 2018.

Etemad, M., Aazam, M., and St-Hilaire, M. (2017). Using devs for modeling and simulating a fog computing environment. In *Computing, Networking and Communications (ICNC), 2017 International Conference on*, pages 849–854. IEEE.

Fricker, C., Guillemin, F., Robert, P., and Thompson, G. (2016). Analysis of an offloading scheme for data centers in the framework of fog computing. *ACM Transactions on Modeling and Performance Evaluation of Computing Systems (TOMPECS)*, 1(4):16.

Gupta, H., Dastjerdi, A. V., Ghosh, S. K., and Buyya, R. (2016). ifogsim: A toolkit for modeling and simulation of resource management techniques in internet of things, edge and fog computing environments. *arXiv preprint arXiv:1606.02007*.

Hong, H.-J., Chuang, J.-C., and Hsu, C.-H. (2016). Animation rendering on multimedia fog computing platforms. In *Cloud Computing Technology and Science (CloudCom), 2016 IEEE International Conference on*, pages 336–343. IEEE.

Hu, Y. C., Patel, M., Sabella, D., Sprecher, N., and Young, V. (2015). Mobile edge computing-a key technology towards 5g. *ETSI White Paper*, 11.

Liu, L., Chang, Z., Guo, X., and Ristaniemi, T. (2017). Multi-objective optimization for computation offloading in mobile-edge computing. In *Computers and Communications (ISCC), 2017 IEEE Symposium on*, pages 832–837. IEEE.

Lopes, M. M., Higashino, W. A., Capretz, M. A., and Bittencourt, L. F. (2017). Myifogsim: A simulator for virtual machine migration in fog computing. In *Companion Proceedings of the10th International Conference on Utility and Cloud Computing*, pages 47–52. ACM.

Mouradian, C., Naboulsi, D., Yangui, S., Glitho, R. H., Morrow, M. J., and Polakos, P. A. (2017). A comprehensive survey on fog computing: State-of-the-art and research challenges. *IEEE Communications Surveys & Tutorials*.

Ongaro, D. and Ousterhout, J. K. (2014). In search of an understandable consensus algorithm. In *USENIX Annual Technical Conference*, pages 305–319.

OpenFog (2017a). Consortium home page. Disponível em: http://www.openfogconsortium.org/. Acesso em: 20 set. 2017.

OpenFog (2017b). Reference architecture for fog computing. Disponível em: http://www.openfogconsortium.org/ra/. Acesso em: 20 set. 2017.

Prazeres, C. and Serrano, M. (2016). Soft-iot: Self-organizing fog of things. In *Advanced Information Networking and Applications Workshops (WAINA), 2016 30th International Conference on*, pages 803–808. IEEE.

Sanchez, L., Muñoz, L., Galache, J. A., Sotres, P., Santana, J. R., Gutierrez, V., Ramdhany, R., Gluhak, A., Krco, S., Theodoridis, E., et al. (2014). Smartsantander: Iot experimentation over a smart city testbed. *Computer Networks*, 61:217–238.

Sarkar, S. and Misra, S. (2016). Theoretical modelling of fog computing: a green computing paradigm to support iot applications. *Iet Networks*, 5(2):23–29.

Satyanarayanan, M. (2017). The emergence of edge computing. *Computer*, 50(1):30–39.

Sonmez, C., Ozgovde, A., and Ersoy, C. (2017). Performance evaluation of single-tier and two-tier cloudlet assisted applications. In *Communications Workshops (ICC Workshops), 2017 IEEE International Conference on*, pages 302–307. IEEE.

Villari, M., Celesti, A., Tricomi, G., Galletta, A., and Fazio, M. (2017). Deployment orchestration of microservices with geographical constraints for edge computing. In *Computers and Communications (ISCC), 2017 IEEE Symposium on*, pages 633–638. IEEE.

Yi, S., Hao, Z., Qin, Z., and Li, Q. (2015). Fog computing: Platform and applications. In *Hot Topics in Web Systems and Technologies (HotWeb), 2015 Third IEEE Workshop on*, pages 73–78. IEEE.