Káio César Ferreira Simonassi

SDMan: um Framework de Gerenciamento Definido por Software

Vitória, ES

2019

Káio César Ferreira Simonassi

SDMan: um Framework de Gerenciamento Definido por Software

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Informática da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do Grau de Mestre em Informática.

Universidade Federal do Espírito Santo – UFES

Centro Tecnológico

Programa de Pós-Graduação em Informática

Orientador: Prof. Dr. Maxwell E. Monteiro

Coorientador: Prof. Dr. Rodolfo da Silva Villaça

Vitória, ES

2019

Káio César Ferreira Simonassi

SDMan: um Framework de Gerenciamento Definido por Software/ Káio César Ferreira Simonassi. – Vitória, ES, 2019-

66 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Dr. Maxwell E. Monteiro

Dissertação de Mestrado – Universidade Federal do Espírito Santo – UFES
Centro Tecnológico
Programa de Pós-Graduação em Informática, 2019.

1. Software Defined Infrastructure. 2. Software Defined Networks. I. Simonassi, Káio César Ferreira. II. Universidade Federal do Espírito Santo. IV. SDMan: um Framework de Gerenciamento Definido por Software

CDU 02:141:005.7

Káio César Ferreira Simonassi

SDMan: um Framework de Gerenciamento Definido por Software

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Informática da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do Grau de Mestre em Informática.

Trabalho aprovado. Vitória, ES, 26 de setembro de 2019:

Prof. Dr. Maxwell E. Monteiro
Orientador

Prof. Dr. Rodolfo da Silva Villaç
Coorientador

Prof. Dr. Magnos Martinello
Membro Interno

Prof. Dr. Gilmar Luiz Vassoler
Membro Externo

Vitória, ES
2019

Agradecimentos

A presente dissertação de mestrado jamais teria acontecido sem o apoio e boa vontade de todos que me cercam.

Não poderia deixar de agradecer meu orientador, Professor Doutor Maxwell Eduardo Monteiro, por sua disponibilidade, paciência, conhecimento e sentido prático com que me orientou neste trabalho e em todos aqueles realizados durante o mestrado.

Agradeço ao meu coorientador, Professor Doutor Rodolfo da Silva Villaça, que contribuiu ativamente com a pesquisa e também sempre me foi muito solicito.

Também agradeço ao Professor Doutor Magnos Martinello por sua ajuda inestimável, bem como a de meus colegas de mestrado e laboratório.

Por fim, quero agradecer à minha família pelo apoio incondicional e suportar toda minha caminhada.

Resumo

No âmbito do Software Defined Infrastructure (SDI) o gerenciamento de redes e serviços foi desconsiderado como um dos principais temas, e a infraestrutura definida por software tem sido gerenciada por soluções de gerenciamento semi-definidas por software. Consequentemente surgiram lacunas na gestão do SDI, como a falta de programabilidade de gerenciamento. Como contribuição tecnológica para reduzir esta e outras lacunas foi projetado e prototipado um framework para programabilidade da gerência de nuvens, batizado de SDMan (Software Defined Management).

Palavras-chaves: Software Defined Infrastructure. Software Defined Networks. Network Management. Cloud Computing.

Abstract

Under Software Defined Infrastructure (SDI), network and service management was disregarded as one of the key themes, and software-defined infrastructure has been managed by software semi-defined management solutions. As a result, there were gaps in SDI management, such as lack of management programmability. As a technological contribution to reduce this and other shortcomings, a cloud management programmability framework, called Software Defined Management (SDMan), was designed and prototyped.

Keywords: Software Defined Infrastructure. Software Defined Networks. Network Management. Cloud Computing.

Lista de ilustrações

Figura 1 – Esquema da Base de Dados do Nova (OPENSTACK) 27
Figura 2 – ClusterControl: Exemplo de Topologia 30
Figura 3 – SDMan Vs. Outras ferramentas 34
Figura 4 – Arquitetura SDMan 38
Figura 5 – Diagrama de sequência do SDMan 40
Figura 6 – Áreas primárias da M-API 43
Figura 7 – Modelo SDMan com replicação 45
Figura 8 – Modelo SDMan fragmentado através das nuvens 46
Figura 9 – SDMan - consumo de recursos 48
Figura 10 – Gráfico gerado através do Neo4J do controller SDMan e suas conexões 49
Figura 11 – SDMan - consumo de recursos 2 49
Figura 12 – Modelo do *Factory Method* para o Connector 52
Figura 13 – VMFs fazendo uso do ProxySQL 57
Figura 14 – Google Cloud: Utilização da CPU 59
Figura 15 – AWS EC2: Utilização da CPU 60
Figura 16 – OpenStack: uso de RAM provido pelo Grafana 60

Lista de abreviaturas e siglas

SaaS	Software as a Service/Software como Serviço
PaaS	Platform as a Service/Plataforma como Serviço
IaaS	Infrastructure as a Service/Infraestrutura como Serviço
SDI	Infraestrutura Definida por Software
ICT	Tecnologia de Informação e Comunicação
API	Interface de Programação de Aplicativos
VIM	Virtualized Infrastructure Management
VNMF	Virtualized Infrastructure Management
SDN	Software DefinedNetwork
NFV	Network Function Virtualization
SDDC	Software Defined Data Center
ETSI	European Telecommunication Standards Institute
MANO	Management and Orchestration
VNF	Virtualized Network Function
OPNFV	Open Platform for NFV
MEF	Metro Ethernet Forum
DMTF	Distributed Management Task Force
IETF	Internet Engineering Task Force
OSM	MANO Open Source
SDMan	Software Defined Management
ManApp	Aplicativos de Gerenciamento
SDManCtrl	Controlador de Gerenciamento do SDMan
M-API	Interface de Programação de Aplicativos de Gerenciamento

SDManProt	Protocolos do SDMan
Conn	Connectors
CPU	Central Process Unit
RDFS	Resource Description Framework Schema
OWL-DL	Ontology Web Language DL
NbInt	Northbound Interface
SbInt	Southbound Interface
EbInt	Eastbound Interface
FCAPS	Fault, Configuration, Accounting, Performance, Security
ITU-T	Telecommunication Standardization Sector
OSI	Open Systems Interconnection
DDB	Distributed Database
IoT	Internet of Things

Sumário

1 INTRODUÇÃO 17
1.1 Objetivos 18
1.1.1 Estrutura do texto 19

2 REVISÃO DA LITERATURA 21
2.1 Computação em Nuvem 21
2.2 Virtualização de Funções de Funções de Rede 23
2.3 A Infraestrutura Definida por Software e a necessidade de orquestração de Nuvens 23
2.4 Desafios e lacunas da gestão da SDI 24
2.5 OpenStack 26
2.6 OSGi 27
2.7 ProxySQL 28
2.8 ClusterControl 29

3 TRABALHOS RELACIONADOS 33
3.1 Comparando as soluções existentes no mercado 33
3.2 As Principais Contribuições do SDMan 34

4 SOFTWARE DEFINED MANAGEMENT (SDMAN) FRAMEWORK 35
4.1 Principais Características 35
4.2 Arquitetura 36
4.2.1 Subcomponentes do SDManCtrl 37
4.2.2 Interfaces de Fronteira 39
4.2.3 Management Application Program Interface (M-API) 39
4.2.4 SDManProt e Primitivas de Recursos de Orquestração - Southbound Interface (SbInt) 41
4.2.5 Interação de Framework de Nuvem Gerenciada - Eastbound Interface (EbInt) 42
4.2.6 Modelo de Distribuição dos Componentes SDMan 44
4.2.7 SDMan em relação a perspectiva da computação de Fog e Edge 46
4.2.8 SDMan Overhead 47

5 IMPLEMENTAÇÃO DE REFERÊNCIA DO SDMAN 51
5.1 Implementação da interface Northbound (NbInt) 51
5.2 Implementação da Interface Eastbound (EbInt) 51
5.3 Implementação do Repositório de Persistência 56

5.4 Integração do SDMan com plataformas de nuvem 57

6 CONCLUSÃO 61
6.1 Considerações Finais 61
6.2 Trabalhos Futuros 61

REFERÊNCIAS 63

1 Introdução

Nos últimos anos a utilização da computação em nuvem vem aumentando em virtude da vantagem da mudança dos custos do cliente em capital para gastos em operação, aliado ao conceito de tempo zero para adquirir (ou liberar) a infraestrutura, plataforma ou serviços de software(KANDIRAJU et al., 2014)(MELL; GRANCE, 2011) e também através de um modelo de negócios no qual o cliente paga somente pelo valor do serviço que foi utilizado (pay-as-you-go-model)(KANDIRAJU et al., 2014)(ALASHOOR, 2014). O aumento da demanda pela computação em nuvem trouxe um novo conjunto completo de desafios à computação em nuvem baseada em data center. Como resposta, a nuvem se espalhou para fora do data center, inaugurando a era da FOG Computing (VAQUERO; RODERO-MERINO, 2014) e da Edge Computing (SHI et al., 2016).

Uma implicação importante dessa descentralização de recursos é o aumento da complexidade da operação. A solução tradicional de gerenciamento em nuvem foi criada com a orquestração e otimização de recursos em um ou alguns sites de computação em nuvem baseados em data center. Mesmo com os enormes avanços proporcionados pela virtualização e pela infra-estrutura definida por software (SDI) (KANDIRAJU et al., 2014) (JARARWEH et al., 2016) o novo cenário envolvendo datacenters em nuvem, públicos e privados, ao longo de cloudlets (SATYANARAYANAN et al., 2009) com equipamentos de ponta, são muito mais complexos do que as soluções atuais de gerenciamento em nuvem são capazes de lidar, onde reações automatizadas, simplesmente baseadas em limites, podem ser potencialmente inúteis neste cenário descentralizado.

O novo cenário trouxe consigo novas soluções, como o Openstack (OPENSTACK, 2019), que no contexto de computação em nuvem, é capaz de atuar oferecendo Infraestrutura como Serviço (IaaS), entretanto, o OpenStack não é capaz de armazenar qualquer informação a respeito das aplicações instaladas numa instância virtualizada. Por outro lado, tratando-se de monitoramento fornecido por ferramentas externas, a título de exemplo, a ferramenta Nagios, é capaz de definir um intervalo de verificação de recurso para saber se um determinado serviço está disponível. Mas infelizmente isso é tudo o que é possível conseguir sobre este serviço, assim as informações que são passíveis de serem recuperadas são insuficientes tanto para o provedor do serviço quanto para o próprio usuário, que podem precisar de mais detalhes das aplicações que não sejam somente sobre sua disponibilidade.

Os pontos levantados a respeito do Openstack e também do Nagios permitiram identificar lacunas da gestão da Infraestrutura Definida por Software (SDI) que precisam ser solucionadas. Para a identificação destas lacunas, foram consideradas as disciplinas clássicas de Tecnologia de Comunicação e Informação (ICT), como segurança da informação,

armazenamento de dados e desenvolvimento de software, uma vez que as mesmas se estabeleceram por conta própria para reivindicar um papel proeminente na SDI. Embora a Gestão de Redes e Serviços tenha sido considerada, ela parece não ter sido abordada como relevante no cenário de computação em nuvem até o momento. As seguintes questões podem estar por trás disto: soluções de gerenciamento são espontaneamente baseadas em software e naturalmente aderem ao universo definido por software, ou outra hipótese, em que o orquestrador e o controlador de software se sobrepuseram a alguns serviços historicamente fornecidos pelo gerenciamento de sistemas.

Embora a comunidade de pesquisa tenha exaltado novos desafios de gerenciamento de SDI na era da nuvem híbrida (WICKBOLDT et al., 2015) (MIJUMBI et al., 2016) e os organismos de padronização definirem as bases para o gerenciamento como ETSI/MANO (ETSI-MANO, 2017), o que tem sido feito até agora é adaptar as soluções tradicionais de gerenciamento à nova realidade (SOFTWARE-DEFINED. . . , 2017). O resultado deste movimento é que a SDI foi gerenciada por soluções semi-definidas por software, isto é, soluções que são atreladas/dependentes de arquitetura que possivelmente resolvem um problema de forma específica e não de maneira geral, trazendo à tona problemas nos campos de estratégia, inteligência de automação e integração, que culminam numa das grandes lacunas da gestão da SDI, a incapacidade de aceitar instruções diferentes daquelas previamente definidas (falta de programabilidade).

Considerando os problemas relacionados a gestão da SDI citados até aqui, este trabalho pretende propor um framework de gerência de estruturas SDI chamado de Software Defined Management (SDMan), de maneira a ser uma solução relevante para melhorar a operação da nuvem. Para tanto, o SDMan é pensado para ser capaz de gerenciar nuvens híbridas (com sites públicos e privados), orquestrar sondas de gerenciamento, testadores e dados gerenciados em diferentes nuvens, permitindo realizar gerenciamento através de diferentes pontos de vista do sistema gerenciado e potencialmente auxiliar nas tomadas de decisão envolvendo volume de recursos e custos de operação.

1.1 Objetivos

O objetivo geral deste trabalho é projetar um framework para programabilidade da gerência de estruturas SDI programáveis (definidas por software). Este objetivo é dividido entre os seguintes objetivos específicos:

- Levantar os principais desafios relacionados a gestão SDI.
- Projetar a arquitetura de um framework de Gerenciamento Definido por Software.
- Prototipar a Implementação de referência do SDMan

1.1.1 Estrutura do texto

A partir deste, o trabalho possui a seguinte organização:

- O Capítulo 2 aborda temas estruturais importantes para a construção deste trabalho.
- O Capítulo 3 trata dos trabalhos relacionados e também posiciona o trabalho na literatura de interesse.
- O Capítulo 4 Apresenta proposta deste trabalho.
- O Capítulo 5 Apresenta implementação de referência deste trabalho.
- O Capítulo 6 Apresenta as conclusões do trabalho bem como trabalhos futuros.

2 Revisão da Literatura

Este capítulo introduz conceitos que embasam o desenvolvimento deste trabalho.

2.1 Computação em Nuvem

A computação em nuvem é um modelo para permitir acesso onipresente de rede sob demanda a um conjunto compartilhado de recursos de computação configuráveis que podem ser rapidamente provisionados e liberados com o mínimo esforço de gerenciamento ou interação do provedor de serviços (MELL; GRANCE, 2011). Este modelo de nuvem é possui os seguintes aspectos listados abaixo:

1. **Características:**

   a) **Serviço sob demanda**. Um consumidor pode provisionar recursos de computação unilateralmente, como tempo do servidor e armazenamento em rede, automaticamente, sem exigir interação humana com cada provedor de serviços.

   b) **Amplo acesso à rede**. Os recursos devem estar disponíveis na rede e acessados por meio de mecanismos padrão que promovem o uso por plataformas heterogêneas.

   c) **Agrupamento de recursos**. Os recursos de computação do provedor são agrupados para atender vários consumidores usando um modelo de multilocatário, com diferentes recursos físicos e virtuais dinamicamente atribuídos e reatribuídos de acordo com a demanda do consumidor.

   d) **Rápida escalabilidade**. Os recursos podem ser provisionados elasticamente e liberados, em alguns casos automaticamente, para escalar rapidamente para fora e para dentro, de acordo com a demanda.

   e) **Monitoramento de serviço**. Os sistemas em nuvem controlam e otimizam automaticamente o uso de recursos, aproveitando um recurso de monitoramento em algum nível de abstração apropriado ao tipo de serviço (por exemplo, armazenamento, processamento, largura de banda e contas de usuário ativas).

2. **Modelos de serviço:**

   a) **Software como serviço (SaaS)**. É fornecer ao consumidor a capacidade de usar os aplicativos do provedor em execução em uma infraestrutura de nuvem. Os aplicativos são acessíveis a partir de vários dispositivos clientes, como um navegador da web ou uma interface de programa. O consumidor não gerencia

ou controla a infraestrutura de nuvem subjacente, incluindo rede, servidores, sistemas operacionais, armazenamento ou até mesmo recursos de aplicativos individuais.

b) **Plataforma como serviço (PaaS)**. É fornecer ao consumidor a capacidade de implantar na infraestrutura de nuvem os aplicativos criados ou adquiridos pelo próprio, criados usando linguagens de programação, bibliotecas, serviços e ferramentas suportadas pelo provedor. O consumidor não gerencia ou controla a infraestrutura de nuvem subjacente, incluindo rede, servidores, sistemas operacionais ou armazenamento, mas tem controle sobre os aplicativos implantados.

c) **Infraestrutura como serviço (IaaS)**. É fornecer ao consumidor capacidade de processamento, armazenamento, redes e outros recursos de computação fundamentais, nos quais o consumidor pode implantar e executar software arbitrário, que pode incluir sistemas operacionais e aplicativos. O consumidor não gerencia ou controla a infraestrutura de nuvem subjacente, mas tem controle sobre sistemas operacionais, armazenamento e aplicativos implantados.

3. **Modelos de implantação:**

a) **Nuvem privada**. A infraestrutura de nuvem é provisionada para uso exclusivo por uma única organização, composta por vários consumidores. Pode ser uma solução proprietária gerenciada e operada pela organização, por terceiros ou por uma combinação deles, e pode existir dentro ou fora das instalações

b) **Nuvem comunitária**. A infraestrutura de nuvem é provisionada para uso exclusivo por uma comunidade específica de consumidores de organizações que compartilham interesses. Pode ser de propriedade, gerenciado e operado por uma ou mais organizações da comunidade, por terceiros ou por uma combinação delas, e pode existir dentro ou fora das instalações

c) **Nuvem pública**. A infraestrutura de nuvem é provisionada para uso aberto pelo público em geral. Pode ser de propriedade, gerenciado e operado por uma empresa, organização acadêmica ou governamental, ou alguma combinação deles. Existe nas instalações do provedor de nuvem

d) **Nuvem híbrida**. A infraestrutura de nuvem é uma composição de duas ou mais infraestruturas de nuvem distintas (privadas, comunitárias ou públicas) que permanecem entidades únicas, mas são unidas por tecnologia padronizada ou proprietária que permite a portabilidade de dados e aplicativos.

2.2 Virtualização de Funções de Funções de Rede

A inserção de novos serviços nas redes atuais tornou-se cada vez mais difícil devido à natureza proprietária dos dispositivos de hardware existentes, ao custo de oferecer espaço e energia para uma variedade de soluções de hardware intermediárias e à falta de profissionais qualificados para integrar e manter estes serviços. A virtualização de funções de rede (NFV) foi proposta para aliviar estes problemas, juntamente com outras tecnologias emergentes, como redes definidas por software (SDN) e computação em nuvem (HAN et al., 2015a).

Os serviços de rede existentes são suportados por diversas funções de rede conectadas de maneira estática. O NFV permite esquemas dinâmicos adicionais para criar e gerenciar funções de rede. Seu conceito principal é o encaminhamento de uma função de rede virtualizada (VNF), que simplifica o provisionamento da cadeia de serviços, criando, modificando e removendo cadeias de serviços de maneira rápida e barata (HAN et al., 2015a). O NFV propõe três características principais:

1. **Separa o software do hardware**. Essa separação permite que o software evolua independentemente do hardware, sendo a recíproca também verdadeira.

2. **Flexibilidade para implantação de funções de rede**. O NFV pode implantar automaticamente o software de funções de rede em um conjunto de recursos de hardware que podem executar funções diferentes em momentos diferentes em diferentes data centers.

3. **Provisionamento dinâmico de serviços**. As operadoras de rede podem dimensionar o desempenho da NFV dinamicamente e conforme a necessidade, com um controle fino de granularidade com base nas condições atuais da rede.

2.3 A Infraestrutura Definida por Software e a necessidade de orquestração de Nuvens

A Infraestrutura Definida por Software (SDI) refere-se à infraestrutura que se transforma de maneira contínua, explorando adequadamente recursos heterogêneos, usando informações obtidas através de monitoramento profundo, para honrar também continuamente os contratos de nível de serviço (SLAs) em face a restrições de fornecedores (KANDIRAJU et al., 2014). Surgiu da comunidade de computação em nuvem após a adoção de tecnologias de virtualização de larga escala, como máquinas virtuais (VMs), SDN (Software Defined Network) e NFV (Network Function Virtualization). Como os data centers geralmente são a plataforma de suporte para computação em nuvem, os ter-

mos: SDDC (Software Defined Data Center), Virtual Data Center, Virtual Infrastructure, Virtual Cloud e Software Defined Infrastructure são considerados sinônimos.

Os padrões NFV do European Telecommunication Standards Institute (ETSI) (ETSI-MANO, 2017) é uma das marcas mais importantes alcançadas pela SDI. Estes documentos definem o relacionamento entre a tecnologia NFV e uma infraestrutura virtualizada típica. Projetado claramente para a comunidade de telecomunicações, os padrões do ETSI trouxeram para o foco os requisitos de Gerenciamento e Orquestração (MANO) para as Funções de Rede Virtualizadas (VNF). A gestão e a orquestração são as questões mais importantes na trajetória de desenvolvimento da SDI e foram abordadas por vários órgãos de padronização (OPNFV, MEF, TMForum, DMTF e IETF). Para transformar a MANO em um padrão *de facto*, o OPNFV deu origem ao Projeto Open Orchestrator, e a ETSI a iniciativa MANO Open Source (OSM). Até hoje, ambos os projetos enfatizam o aspecto de orquestração da MANO, claramente a grande angústia entre os operadores de nuvem e data center.

Algumas vozes dissidentes argumentaram que o MANO não é aplicável a nuvens fora do contexto da telecomunicação (CHAPPELL, 2015). No entanto, o MANO tem semelhanças demais com o modelo tradicional de IaaS, PaaS e SaaS, especialmente quando a nuvem suporta contêineres virtuais e arquiteturas de micro-serviços. Com relação a essas limitações, neste trabalho optou-se por considerar a SDI como uma infraestrutura genérica, virtualizada e definida por software, para qualquer arquitetura lógica e tecnologias de virtualização.

Embora as tradicionais plataformas cloud view, como OpenStack, OpenNebula, VMWare vOneCloud e Apache Mesos, forneçam suporte para gerenciamento, automação e operação são notórias as lacunas na orquestração e gerenciamento de SaaS (CHAPPELL, 2015). Essas lacunas são parcialmente preenchidas por ferramentas de terceiros, como Puppet, Chef, Salt e Kubernetes. Este ecossistema composto por todas essas ferramentas se tornou o padrão *de fato* para muitos operadores de nuvem. Algumas iniciativas como o OpenStack Tacker pretendem apoiar o paradigma NFV em plataformas de nuvem tradicionais. Infelizmente, essas iniciativas estão em sua infância e seus serviços de gerenciamento dependem totalmente dos serviços de gerenciamento de plataformas em nuvem e estão sujeitos às mesmas lacunas.

2.4 Desafios e lacunas da gestão da SDI

Apesar de a maioria dos desafios gerais identificados em (HAN et al., 2015b), (WICKBOLDT et al., 2015), (MIJUMBI et al., 2016), (SOFTWARE-DEFINED..., 2017), e (MORENO-VOZMEDIANO; MONTERO; LLORENTE, 2013) estarem de alguma forma relacionados às atividades de gerenciamento, na verdade, não há uma lista explícita de

desafios de gerenciamento na literatura de computação em nuvem. O que os autores enfatizaram como questões relevantes para a próxima geração de nuvem foram: inteligência de orquestração, recursos de auto-gerenciamento e integração de ferramentas. Eles reconhecem que a orquestração, especialmente para a tarefa de automação, foi o principal vetor de desenvolvimento explorado até agora, impondo alguns problemas de gerenciamento:

a) **Falta de uma distinção bem definida entre as funções dos Orquestradores de Nuvem, os controladores de rede e o processo de gerenciamento tradicional.** Todos eles são sobrepostos de tal forma que não há um limite claro entre tarefas de automação, orquestração e gerenciamento tradicional, embora funções de gerenciamento clássicas como provisionamento, faturamento, segurança e configuração ainda estejam presentes. Assim, se alguma tarefa complexa de gerenciamento precisar ser feita: (i) o orquestrador estará sobrecarregado; (ii) uma ferramenta externa (não integrada) será usada para a tarefa. O monitoramento é a única tarefa de gerenciamento reconhecida. E, por ora, os orquestradores e controladores devem agir em resposta a alguma condição de nuvem, eles executam o monitoramento por si mesmos. Como alguns dados monitorados podem ser úteis para várias finalidades, *este tipo de tarefa de gerenciamento deve ser realizado por um serviço integrado e compartilhado*;

b) **Falta de programabilidade de gerenciamento, que forneça alguma inteligência de automação.** Essa inteligência pode ser obtida de diferentes perspectivas. Mas, a correlação entre informações produzidas por diferentes ferramentas de gerenciamento é um dos instrumentos mais consistentes para fornecê-las. *Um framework de gerenciamento programável pode permitir o uso de paradigmas de gerenciamento mais complexos*, como: gerenciamento baseado em aprendizado de máquina e gerenciamento autônomo baseado em políticas. As ferramentas de automação de gerenciamento de nuvem baseadas em script não fornecem inteligência de automação sofisticada, já que os comandos de script são limitados pela API de gerenciamento de nuvem.

c) **Ausência de orquestração de recursos de Gerenciamento Explícito.** As sondas de gerenciamento, bancos de dados, processo e ferramentas de teste de carga são operados como um domínio separado dentro da nuvem. *Não há um serviço explícito de "orquestração de gerenciamento"que imponha padrões de gerenciamento e limite o desenvolvimento de novas abordagens de gerenciamento.*

d) **Ecossistema de Gerenciamento Deficiente e integração de informações.** A integração entre ferramentas de gerenciamento independentes e a plataforma de computação em nuvem não foi alcançada em todo o seu potencial. As plataformas de nuvem têm suas próprias ferramentas básicas de gerenciamento e repositório de dados. Não existe um mecanismo de integração explícito para acomodar ferra-

mentas de gerenciamento externas e isso força o uso de soluções circunstanciais, geralmente baseadas em integração de dados sintáticos (implícitos). Infelizmente, a falta de semântica de informações de gerenciamento é uma grande barreira para a integração completa de informações na nuvem, fator que poderia impulsionar a inteligência de automação. Por exemplo, informações de gerenciamento ainda estão sendo armazenadas em bancos de dados relacionais tradicionais com semântica implícita. Dessa forma, é muito comum o nascimento de ferramentas especializadas de gerenciamento de nuvem, como uma ferramenta de consultoria de segurança em nuvem. *Essa abordagem tradicional implementa ferramentas independentes como uma solução não integrada*, criando um novo silo de informações de gerenciamento, diminuindo tanto a inteligência de automação em potencial por meio de programação quanto a orquestração explícita de recursos de gerenciamento.

e) **Falta de extensão de recursos de gerenciamento de plataforma em nuvem.** A maioria dos frameworks de nuvem, como o OpenStack(SEFRAOUI; AISSAOUI; ELEULDJ, 2012) e CloudStack(CLOUDSTACK, 2017), mantém seus dados de máquinas físicas e virtuais em bancos de dados relacionais, mantendo um serviço de monitoramento básico. Dessa forma, um aplicativo de gerenciamento externo precisa conhecer o esquema do banco de dados e todo o relacionamento de dados implícito (por exemplo, chaves estrangeiras de cada tabela) para usar estes dados. Infelizmente, serviços virtualizados e funções de rede, hospedados nas máquinas virtuais, não possuem dados de gerenciamento disponíveis no mesmo banco de dados de gerenciamento. Essa restrição de gerenciamento de nuvem defende que os dados e funções de gerenciamento de nuvem incorporados podem não ser suficientes para lidar com os requisitos modernos de operações complexas na nuvem, por exemplo, a correlação de máquinas físicas e virtualizadas, bem como as funções e serviços virtualizados implantados.

2.5 OpenStack

O OpenStack é uma tecnologia que visa prover Infraestrutura como serviço (IaaS) para a computação em nuvem (SEFRAOUI; AISSAOUI; ELEULDJ, 2012). No entanto avaliando o OpenStack é possível notar que o mesmo não armazena nenhuma informação a respeito das aplicações instaladas numa instância, esta informação pode ser verificada, como pode ser observado na Fig 1, ao fazer uma análise na base de dados de um dos principais serviços da tecnologia, o Nova, que atua como motor da computação primária e tem a função de levantar e gerenciar as instâncias da nuvem.

Figura 1 – Esquema da Base de Dados do Nova (OPENSTACK)

2.6 OSGi

A tecnologia OSGi Alliance (anteriormente conhecida como a initiative Open Services Gateway) objetiva gerenciar o ciclo de vida de componentes de software de qualquer lugar da rede. O OSGi é um conjunto de especificações que definem um sistema de componente dinâmico para Java. Tais especificações permitem um modelo de desenvolvimento onde as aplicações são dinamicamente compostas de diferentes módulos (pacotes). Além

disso a especificação permite que os módulos escondam suas implementações de outros, garantindo a comunicação por meio do conceito de serviços (objetos que são especificamente compartilhados entre os componentes) (ALLIANCE, 2003). Algumas das principais vantagens de se utilizar o OSGi são:

1. **Modularidade**. O código é dividido em pedaços muito pequenos, facilitando a reutilização do código posteriormente.

2. **Baixo tempo de inatividade**. Os contêineres OSGi permitem a parada, atualização, reinicialização etc. de um único pacote, sem interromper a aplicação. Além disso, várias versões do código podem ser executadas ao mesmo tempo no mesmo contêiner.

3. **Carregamento de classe isolado**. É possível carregar versões diferentes de uma mesma biblioteca no contêiner ao mesmo tempo sem causar conflitos de carregamento de classe. As versões de uma mesma biblioteca são configuráveis para cada pacote da aplicação, as bibliotecas são especificadas e carregadas conforme definido nos manifestos individuais dos pacotes, o OSGi garantirá que a versão correta seja fornecida a cada um.

4. **Controle de importação e exportação refinado**. Ao criar um pacote OSGi são definidos quais outros devem ser importados e também quais pacotes o novo módulo exportará como serviço. Pacotes de 'importação' são aqueles que são obtidos de outros pacotes, enquanto pacotes de 'exportação' são aqueles que o pacote desenvolvido disponibiliza para outros pacotes que dependem dele.

2.7 ProxySQL

ProxySQL é um serviço de proxy que adquire solicitações MYSQL de clientes e, em seguida, os transmite para os serviços de backend e, então, encaminha o resultado recebido para o solicitante (ProxySQL,).

Este trabalho é projetado para lidar com diferentes nuvens, sendo capaz de ter seu núcleo executado em uma determinada nuvem (AWS, Gooogle Cloud, por exemplo), mas também de ter seus módulos rodando em outras nuvens e comunicando-se entre si. Esta modularidade implica em aumento de complexidade, a qual faz-se uso do ProxySQL para mitigá-la.

A seguir, são apresentadas as principais características da do ProxySQL que motivam sua utilização:

- ***Query caching***: os resultados podem ser armazenados em cache em um período

de tempo configurável, no formato nativo dos pacotes do MySQL.

- **Suporte a *failover***: embora o ProxySQL não forneça failover de suporte como um recurso, ele colabora suavemente com as ferramentas existentes que o habilitam. Ele monitora a integridade dos backends com os quais se comunica e pode evitá-los temporariamente com base em taxas de erros configuráveis.

- ***Query routing***: para o caso avançado em que diferentes classes de consulta precisam ser roteadas para diferentes clusters do MySQL com diferentes configurações. Com base em um mecanismo de correspondência avançada, é possível rotear de forma transparente as consultas em direção ao cluster de destino que pode executá-las com mais eficiência.

- ***Advanced configuration with 0 downtime***: o sistema de configuração ProxySQL é inspirado nos roteadores. Você pode configurar tudo dinamicamente, persistir a configuração e modificá-la. Tudo com 0 downtime.

- ***Application layer proxy***: o ProxySQL faz o roteamento de tráfego de maneira cega. Ele entende o protocolo MySQL e age de acordo. É por isso que é possível atender com facilidade casos de uso avançados, como transações difíceis ou a geração de estatísticas detalhadas e em tempo real sobre a carga de trabalho.

- ***Advanced topology support***: proxies em cascata para maior disponibilidade e flexibilidade. Suporte para topologias complexas do MySQL, envolvendo replicação e failover ou espelhamento de consulta. Tudo isso feito sem esforço pelo ProxySQL.

- ***Firewall***: no caso de consultas ofensivas que causam problemas ao banco de dados (SQL injection ou recuperação ineficaz de informações via SELECT * sem WHERE, por exemplo), o ProxySQL atua como um gatekeeper entre o aplicativo e o banco de dados, permitindo que os DBAs reajam rapidamente.

2.8 ClusterControl

ClusterControl é um sistema de gerenciamento de infraestrutura de banco de dados de código aberto. A motivação inicial para utilizá-lo deve-se ao fato de trabalhar com diversas tecnologias como MySQL, PostgreSQL, MariaDB, MongoDB, Docker, ProxySQL, entre

outras(CLUSTERCONTROL, 2019). Esta poderosa ferramenta possui diversos recursos interessantes ao problema da persistência com replicação de banco de dados, oferecendo suporte a balanceamento de carga, inclusive utilizando ProxySQL. O ClusterControl permite implementar balanceadores de carga, que é um recurso de alta estima para disponibilidade do banco de dados; especialmente por tornar as alterações de topologia transparentes para aplicativos e implementar a funcionalidade de divisão e leitura de gravação. Na Fig. 2 é mostrada, na interface do ClusterControl, uma topologia de cluster montada (usando virtualização) contendo alguns balanceadores de carga bem como os nós da topologia.

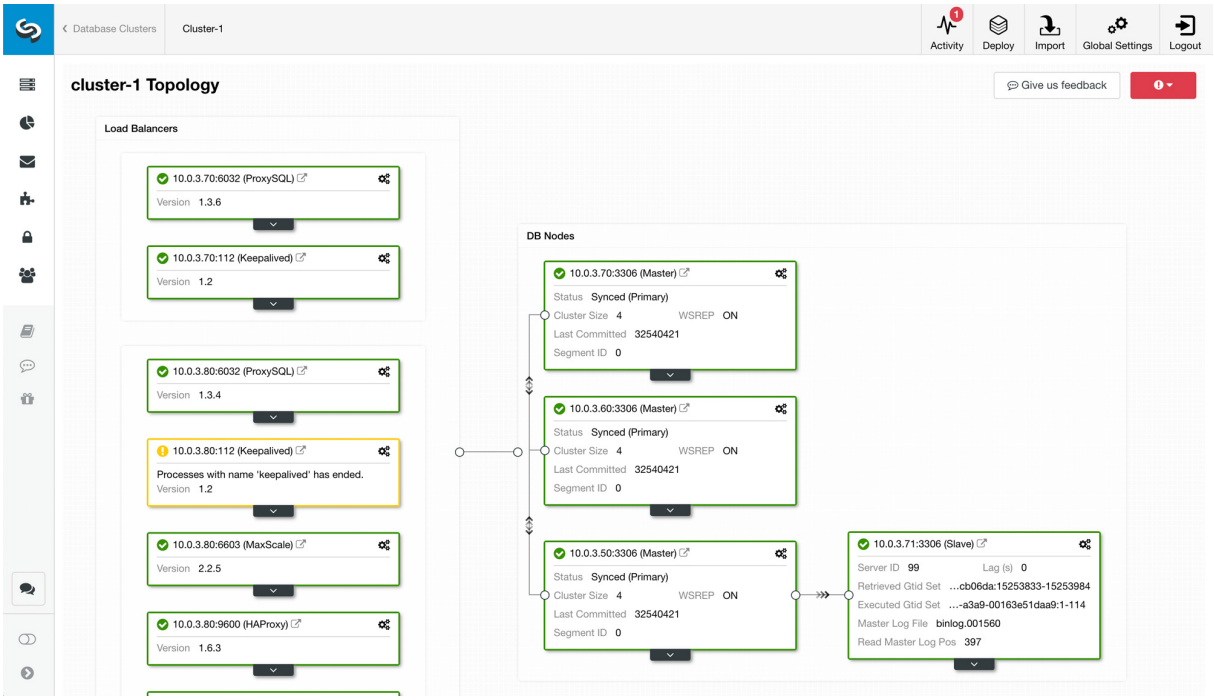

Figura 2 – ClusterControl: Exemplo de Topologia

Outros recursos úteis disponíveis:

- ***Atualizações de Banco de Dados***: garante que os bancos de dados permaneçam atualizados (oferece relatórios de erros caso seja necessário tomar alguma ação), reduzindo o tempo de manutenção, economizando tempo e dinheiro.

- ***Gerenciamento de Backup Centralizado***: fornece backups de grandes conjuntos de dados, recuperação pontual, criptografia de dados em trânsito, integridade dos dados por meio de verificação automática de restauração, backup na nuvem (AWS, Google e Azure) para recuperação de desastres, políticas de retenção e alertas, além de relatórios.

- ***Monitoramento de Banco de Dados em Tempo Real***: oferece uma visão unificada de todas as implantações de banco de dados em vários datacenters, permi-

tindo visualizar a imagem geral ou detalhando os nós individuais, tudo em tempo real.

- ***Relatório Operacional do Banco de Dados***: relatórios operacionais abrangentes sobre o banco de dados envolvendo informações sobre sua integridade e a estabilidade das operações.

- ***Recuperação e Reparo automatizados do Banco de Dados***: em caso de falha o ClusterControl mantém automaticamente os clusters de banco de dados, reparando automaticamente caso sejam nós que utilizam Galera Cluster, ou através da promoção de um nó escravo para mestre e salvando outras réplicas para o caso de configurações de replicação assíncrona.

3 Trabalhos Relacionados

Este capítulo apresenta os trabalhos relacionados e destaca pontos chave da contribuição desta dissertação.

3.1 Comparando as soluções existentes no mercado

A maioria dos provedores de nuvem usa suas próprias ferramentas de monitoramento proprietárias e costumeiramente dependentes de plataforma. Por exemplo, o OpenStack Fuel(OPENSTACKFUEL, 2017) gerencia o OpenStack, o Amazon Cloud Watch(AMAZON, 2017) monitora o Amazon Web Services (AWS) e o CloudMonix (antigo Azure Watch)(CLOUDMONIX, 2015) supervisiona recursos azure-based. Por outro lado, ferramentas de monitoramento de nuvem independentes de plataforma, como RightScale (Rightscale,), Unified Infrastructure Management (anteriormente Nimsoft)(TECHNOLOGIES, 2017), Monitis(MONITIS, 2017) e Rackspace(RACKSPACE, 2017) podem ser utilizadas para monitorar uma multiplicidade de plataformas de nuvem. Algumas dessas ferramentas de monitoramento de nuvem independentes de plataforma foram projetadas inicialmente para monitorar a infraestrutura de TI e, posteriormente, foram estendidas para monitorar os níveis de infraestrutura e de aplicativos(FATEMA et al., 2014). No entanto, algumas ferramentas podem monitorar apenas a infraestrutura ou a nível de aplicação.

Plataformas de código aberto também estão disponíveis no mercado, como o Nagios (NAGIOS, 2017), o OpenNebula (OPENNEBULA.ORG, 2017) e o CloudStack. O Nagios é uma plataforma de monitoramento de código aberto bem conhecida que foi estendida para suportar o monitoramento de infraestruturas em nuvem. O OpenNebula é um kit de ferramentas de código aberto para o gerenciamento de infraestruturas de nuvem distribuídas e heterogêneas. O CloudStack, por sua vez, foi projetado para implantar e gerenciar grandes redes de máquinas virtuais, como uma plataforma em nuvem altamente disponível e escalável. No entanto, nenhuma dessas ferramentas de monitoramento de nuvem aberta e independente foi projetada na era da nuvem, em uma abordagem limpa e simples.

Há também outras tecnologias propostas que visam soluções diferentes para problemas dissonantes em relação aos levantados por neste trabalho, como o SDCM (Software-Defined Cloud Machine) (THAMES; SCHAEFER, 2016) e o SDCon (SON; BUYYA, 2019). O SDCM, por exemplo, é inspirado em redes definidas por software e tem como objetivo propor uma arquitetura para fornecer sistemas de hardware genéricos e reconfiguráveis, abertos e facilmente modificáveis no nível do software. SDCon é um Cloud Scheduler em que os mecanismos de monitoramento e gerenciamento estão amarrados aos do Open

Daylight e OpenStack.

A imagem 3 mostra que a maioria das ferramentas de gerenciamento de nuvem atuais não superou uma ou mais lacunas de gerenciamento listados na seção 2.4. O framework SDMan foi projetado para diminuir essas lacunas.

Ferramentas de Gerenciamento	Lacunas Superadas				
	a	b	c	d	e
Rigth Scale	N/A				
Fuel	✓				
Amazon Cloud Watch	✓				
CloudMonix	✓				
OpenNebula	✓				
CloudStack					
Nagios	N/A				
Monitis	N/A				
Rackspace	✓				
SDCM	✓				
SDCon					
SDMan	✓	✓	✓	✓	✓

Figura 3 – SDMan Vs. Outras ferramentas

3.2 As Principais Contribuições do SDMan

Como o SDMan é um framework arquitetural, é difícil medir o quanto ele pode melhorar a operação da nuvem. No entanto, seus recursos podem suportar a extensão real do recurso de gerenciamento de nuvem. Pensando em um cenário no qual uma organização tem que gerenciar uma nuvem híbrida (com sites públicos e privados), o SDMan é capaz de orquestrar probes(sondas) de gerenciamento, testadores e dados gerenciados em diferentes nuvens, permitindo diferentes pontos de vista de gerenciamento dos sistemas gerenciados. Assim como é o único capaz de integrar dados de diferentes ferramentas de gerenciamento de nuvem, com o potencial de viabilizar uma automação inteligente cruzando os dados obtidos de diferentes nuvens. Cruzando dados de consumo de recurso ao longo do tempo, por exemplo, seria possível mensurar grandezas como o custo de operação e então implementar ações automatizadas para mover recursos de uma nuvem A para uma nuvem B, afim de mitigar despesas.

Considerando estes aspectos, os principais beneficiados ao utilizar a tecnologia proposta pelo SDMan seriam os clientes que utilizam serviços de nuvem em larga escala, a título de exemplo, poderia ser citado o Dropbox, empresa que fornece serviços de computação em nuvem utilizando, atualmente como IaaS, a Amazon Web Services (AWS).

4 Software Defined Management (SDMan) framework

Neste capítulo são apresentadas as principais características do framework: controle e separação de execução, programabilidade, protocolo de interação padrão, serviços de gerenciamento de componentes SDI, orquestração de recursos de gerenciamento, repositório de dados operacionais e semânticos, a arquitetura SDMan e seus subcomponentes.

4.1 Principais Características

Para ser considerado como uma estrutura de Gerenciamento Definido por Software, o SDMan oferece as mesmas abstrações fornecidas por uma solução típica do SDAny. Essas abstrações são as seguintes:

- **Controle Separado da Execução** Os aplicativos de gerenciamento (ManApp) são executados na parte superior de um Controlador de gerenciamento (SDManCtrl). Eles comandam ações para os elementos VMFs (Virtualized Management Functions) que, por sua vez, executam as funções de gerenciamento.
- **Programabilidade**: Aplicativos de gerenciamento são desenvolvidos por programadores usando a Interface de Programação de Aplicativos de Gerenciamento (M-API). As primitivas M-API instruem a interação, via protocolo SDMan (SDManProt), para as funções de gerenciamento virtualizadas (VMFs).
- **Protocolo de interação padrão**: A execução de tarefas de gerenciamento tradicionais pode ser feita por ferramentas pré-existentes e maduras adaptadas como VMFs. O protocolo SDMan (SDManProt) é um conjunto abrangente de regras capaz de (i) orquestrar recursos de computação e rede para suportar VMFs, e (ii) instruir os VMFs sobre como executar suas funções.

No entanto, para atender aos objetivos do framework SDMan é necessário adicionar recursos adicionais, especialmente para fins de integração, como:

- **Serviços de gerenciamento para componentes SDI**: Os controladores e orquestradores SDI podem precisar de informações e ações de gerenciamento externas específicas, bem como oferecer dados de gerenciamento sobre seus recursos e operação.
- **Orquestração de recursos de gerenciamento**: O controlador de gerenciamento

deve organizar e manter recursos de computação e de rede para fornecer o ambiente de execução do SDMan.

- **Repositório de dados operacionais e semânticos**: O controlador de gerenciamento deve organizar os repositórios de dados de gerenciamento considerando os requisitos operacionais, como as demandas de configuração e monitoramento. Além disso, estes dados devem ser correlacionados com descritores semânticos, permitindo o processamento formal e lógico dos dados requeridos por soluções de gerenciamento autônomas e autocontroladas.

4.2 Arquitetura

Na Fig. 4 é apresentada a arquitetura do SDMan e como ele foi projetado para cobrir os recursos propostos nas seções anteriores.

- **Management Application (ManApp)** é o componente de software responsável por um grupo de operações de gerenciamento, desenvolvido usando M-API. Uma operação de gerenciamento bem definida é articulada ao executar uma ou mais funções de gerenciamento virtualizadas em uma ordem fixa.

- **Management Controller (SDManCtrl)** atua como um middleware, que por meio de primitivas M-API executa instruções de aplicativos de gerenciamento. O SDManCtrl também oferece serviços para aplicativos externos, como estruturas de nuvem autônomas gerenciadas e de terceiros. Os subcomponentes do SDManCtrl são: (i) o **Semantic Services Provider**, (ii) o **Topology Manager**, (iii) o **Management Services Supervisor** e (iv) o **Communication Mediator**. Os subcomponentes são detalhados em 4.2.1.

- **Virtualized Management Functions (VMF)** são artigos de software que executam ações de gerenciamento com base em suas finalidades operacionais. Ferramentas como o RRDTools, Nagios e Iperf3 podem ser adaptadas por um *wrapper* de software para interagir com o SDManCtrl através do SDManProt.

- **Connectors (Conn)** são softwares desenvolvidos com base na tecnologia fornecida pela nuvem que atuam como adaptadores tanto na mediação da troca de informações quanto na execução de ações entre a plataforma de nuvem gerenciada e o SDManCtrl. A função dos conectores é solicitar dados de gerenciamento para a nuvem, bem como as ações de comando que devem ser executadas pela nuvem, além de ser responsável por receber solicitações de serviços de gerenciamento de nuvem e enviá-las para o SDManCtrl.

- **Persistence repository**: composto por bancos de dados operacionais e semânticos, é responsável por armazenar os dados de gerenciamento. Bancos de dados operacionais são usados pelo VMF (Virtualized Management Function), geralmente, para armazenar dados históricos de monitoramento e configuração. Os VMFs, iniciados pelo SDManCtrl, podem acessar diretamente o banco de dados operacional designado (um link é estabelecido) sem que o controlador tenha que mediar. O banco de dados semântico, diferentemente do banco operacional, concentra-se na descrição formal dos dados de gerenciamento, de modo que é possível fornecer interação com soluções de gerenciamento inteligentes e autônomas.

4.2.1 Subcomponentes do SDManCtrl

Como mencionado anteriormente, o SDManCtrl possui componentes internos responsáveis por algumas tarefas especiais. O **Multi-layer Topology Manager** é um banco de dados semântico leve responsável por manter as informações sobre a configuração das entidades gerenciadas e o relacionamento entre elas (dependência, parte inteira, etc.). Conforme descrito pelos padrões ETSI/MANO, uma plataforma de nuvem é organizada em camadas nas quais funções virtuais, recursos de infraestrutura virtual e recursos físicos são armazenados. Como cada camada apresenta seus próprios requisitos de gerenciamento, este subcomponente usa uma linguagem de descrição de recursos de computação sugerida em (MONTEIRO et al., 2014).

O **Communication Mediator** é o componente interno responsável pelo roteamento, adaptação e troca de mensagens entre os componentes do SDMan. É um tipo de barramento de comunicação projetado para ocultar a complexidade da troca de mensagens entre entidades internas e externas. Como pode ser lido na sub-seção Interfaces de Fronteira, o SDMan é exposto a várias interfaces de interação, cada uma com uma finalidade específica e, provavelmente, uma tecnologia específica.

O **Management Services Supervisor** é responsável por avaliar as solicitações de serviço de gerenciamento e decidir se uma nova instância de serviço precisa ser criada ou se uma existente pode ser usada. A ideia por trás deste subcomponente é evitar várias instâncias da mesma tarefa de gerenciamento. Um exemplo típico é o caso em que dois ManApps solicitam o mesmo serviço de monitoramento, visando o mesmo objeto gerenciado (carga da CPU, por exemplo). Neste caso, este sub-componente deve iniciar apenas uma tarefa de monitoramento, compartilhando os dados coletados entre estes dois ManApps.

Finalmente, o **Semantic Services Provider** organiza os dados de gerenciamento no repositório semântico e fornece acesso externo a eles. Baseado em tecnologias semânticas, como RDFS, OWL-DL e raciocinadores, este subcomponente é o link com sistemas externos de gerenciamento inteligente e autocontrolados. Ele fornece a descrição formal dos recursos

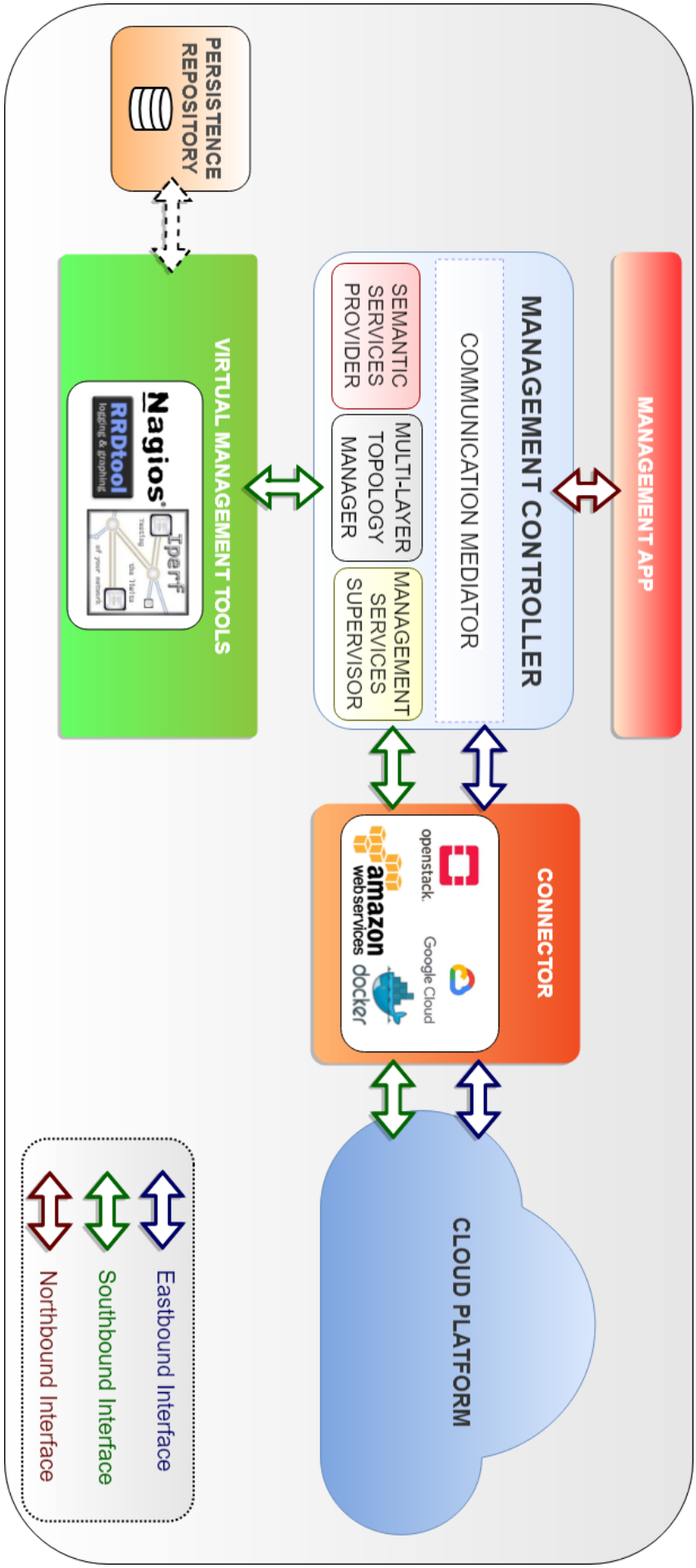

Figura 4 – Arquitetura SDMan

da nuvem e as respectivas informações de gerenciamento.

4.2.2 Interfaces de Fronteira

Como pode ser observado na Fig. 4, o framework SDMan possui quatro interfaces de interação diferentes:

a) **Northbound Interface (NbInt)** é a interface entre os ManApps e o SDManCtrl, implementados pelas primitivas M-API. Essas primitivas são explicadas nas subseções a seguir.

b) **Southbound Interface (SbInt)** é responsável pela interação com os VMFs (SbInt.vmf) e a nuvem gerenciada (SbInt.cloud) na qual os recursos de gerenciamento são alocados. As primitivas SbInt.vmf foram projetadas para configurar como um VMF deve funcionar. O SbInt.cloud, por sua vez, são primitivas projetadas para lidar com a orquestração dos VMFs, usando a mediação fornecida pelos Conectores. Por exemplo, um VMF pode ser alocado em uma máquina virtual do SDMan (exemplo de responsabilidade do SbInt.cloud) e configurado para coletar informações de disponibilidade de um grupo VNF específico na mesma nuvem (exemplo de responsabilidade do SbInt.vmf).

c) **Eastbound Interface** é a interface de interação com a estrutura de nuvem gerenciada, permitindo a troca de dados e mensagens. Considerando que as estruturas de nuvem gerenciadas não alteram suas próprias interfaces, a interface Eastbound precisa de um Conector para mediar a interação com o SDManCtrl.

d) **Westbound Interface** é a interface de interação que fornece acesso externo aos recursos semânticos do SDManCtrl. Entretanto é preciso deixar bem claro que esta interface está fora do escopo deste trabalho.

Fig. 5 Ilustra as interações iniciais entre os componentes do SDMan.

4.2.3 Management Application Program Interface (M-API)

A interface de programação de aplicativos de gerenciamento (M-API) foi concebida usando as funções essenciais de gerenciamento de rede. Conforme descrito por Black (BLACK, 1994), o Gerenciamento de Rede é a tarefa de planejar, organizar, monitorar, contabilizar e controlar atividades e recursos de uma rede.

A Fig. 6 mostra um fragmento de funções e tarefas que compõem o M-API, bem como sua relação com as áreas funcionais tradicionais de gerenciamento OSI/ITU-T X.700 (FCAPS - Fault, Configuration, Accounting, Performance, Security).

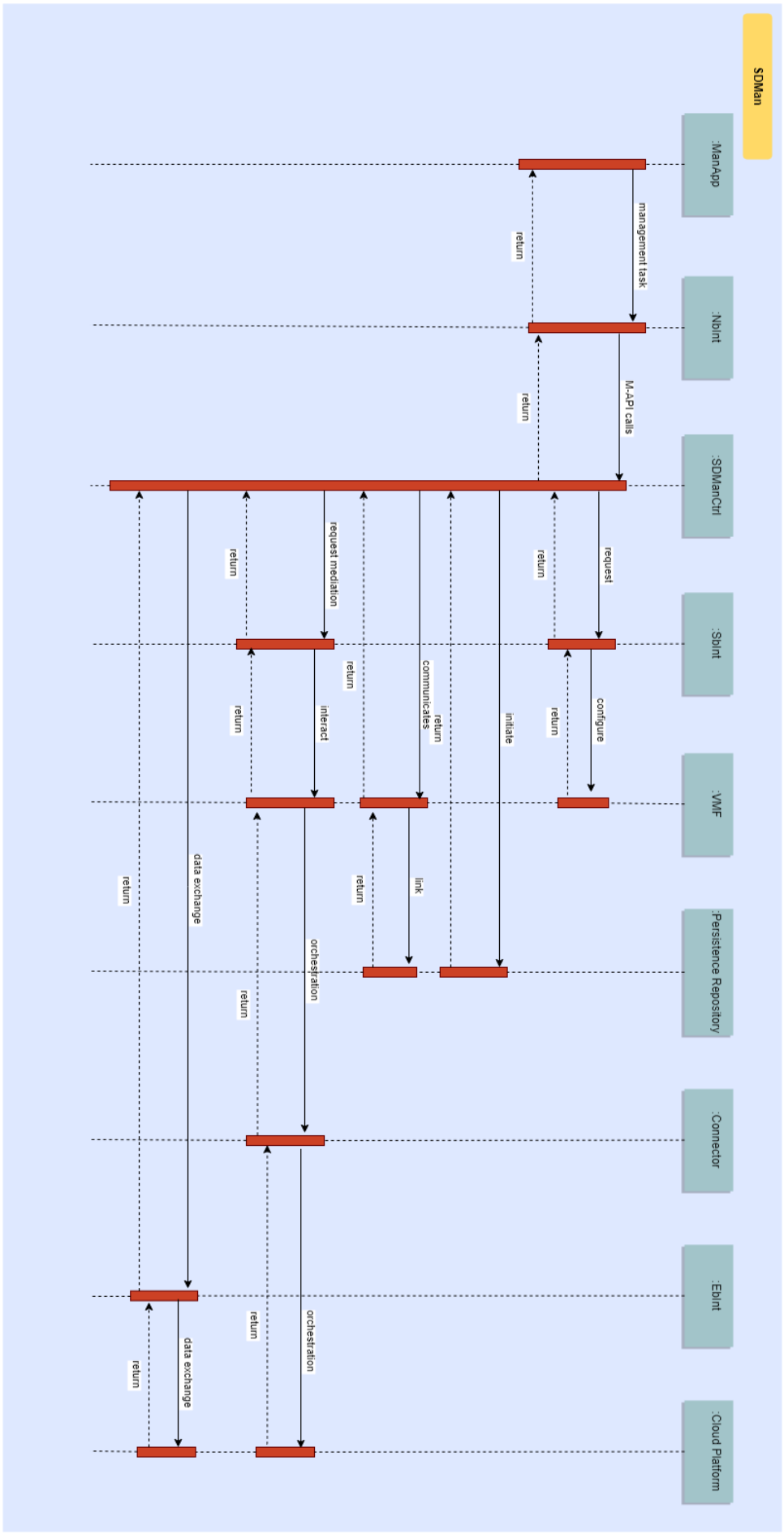

Figura 5 – Diagrama de sequência do SDMan

As primitivas M-API que interagem com o VMF (Virtualized Management Functions) foram projetadas no padrão de software *Abstract Factory*. Essa estratégia de projeto permite o desacoplamento entre o código do aplicativo de gerenciamento e o código específico da função de gerenciamento virtualizado. Cada código de manipulação específico do VMF deve oferecer um *concrete factory method* para implementar o M-API *Abstract Factory*.

Um exemplo dessa estratégia de design pode ser visto na Listagem 4.1, no qual a classe *Monitoring-History Abstract Factory class* é apresentada na notação de Linguagem Java, dentro do pacote de Monitoramento.

Listagem 4.1 – Monitoring-History Abstract Factory

```
1  package sdman.mapi.monitoring
2
3  public abstract class MonHistory{
4        public abstract void setManagedElement();
5        public abstract void setTargetObject();
6        public abstract void setMonitorinFrequency();
7        public abstract void setDataRepository();
8        public abstract int startMonHistory();
9        public abstract int stopMonHistory();
10 }
11
12 public interface AbstractMonHistoryFactory{
13       public MonHistory createMonHistory();
14 }
```

Completando essa estratégia de projeto, apresentamos, na notação da Linguagem Java, um fragmento da classe concreta (CNagiosMonHistory ⇒ que estende MonHistory, descrito acima) e a interface de fábrica (interface NagiosMonHistoryFactory) para o *NagiosVMF*. Os *concrete factory method* estão associados aos pacotes do VMF, como pode ser visto na Listagem 4.2. Estes pacotes devem ser desenvolvidos pelo provedor da ferramenta de gerenciamento.

4.2.4 SDManProt e Primitivas de Recursos de Orquestração - Southbound Interface (SbInt)

Como mencionado anteriormente, a interface Southbound foi dividida em dois grupos primitivos: $SbInt.vmf$ e $SbInt.cloud$. Devido a seus diferentes objetivos, essas subinterfaces podem ser implementadas usando diferentes abordagens de design. O $SbInt.vmf$ está preocupado principalmente com o gerenciamento de configuração do VMF. Assim, para essa função, é conveniente aplicar um protocolo de configuração maduro e extensível, como **Netconf**. O SDManProt.vmf (SDManProt para interação da função de gerenciamento virtualizado) é um perfil YANG focado na configuração do VMF. Infelizmente, não foi desenvolvido nesta fase de trabalho. No entanto, **Netconf** não é capaz de alterar diretamente uma configuração de ferramentas de gerenciamento. Para este trabalho, um

Listagem 4.2 – CNagiosMonHistory

```
1  package sdman.mapi.vmf.nagios
2
3  public class CNagiosMonHistory extends MonHistory{
4      /* attributes here */
5    public CNagiosMonHistory(String manEl, String tObj,String monFreq, String
         dataRepo){
6    }
7    @Override
8    public void setManagedElement(String manEl) {
9      this.managedElement=manEl;
10   }
11   /* other member methods here */
12 }
13
14 public interface NagiosMonHistoryFactory implements AbstractMonHistoryFactory{
15
16     public NagiosMonHistoryFactory (String manEl, String tObj, String monFreq,
           String dataRepo){
17     }
18     @Override
19     public MonHistory createMonHistory(){
20     return new CNagiosMonHistory(manEl,tObj,monFreq,dataRepo);
21     }
```

tipo de *software wrapper* deve existir para ligar as funções da ferramenta de gerenciamento com o serviço **Netconf**.

O *SbInt.cloud* é totalmente dedicado à orquestração do VMF. Considerando o papel desempenhado pelos Connectors (mediadores entre SDManCtrl e a nuvem gerenciada). É conveniente juntar essas primitivas de interface à mesma filosofia da interface Eastbound. Conseqüentemente, o Connector deve traduzir as primitivas SDManProt.cloud (SDMan-Prot para interação na nuvem) para serviços ou interfaces específicos da nuvem. No futuro, outros protocolos de orquestração, como Puppet (PUPPET,) e Salt (CHERRIER et al., 2013), podem ser explorados como opções para essa interface.

4.2.5 Interação de Framework de Nuvem Gerenciada - Eastbound Interface (EbInt)

A interface Eastbound pode ser observada por dois pontos de referência diferentes. O primeiro é o ponto de referência interno, na fronteira entre o SDManCtrl e o Connector ($EbRefPoint_int$). O segundo é o ponto de referência externo, entre o Conector e o framework da nuvem gerenciada ($EbRefPoint_ext$). Para estes dois pontos de referência, as primitivas são as mesmas, mas a tecnologia de implementação pode variar, especialmente, para as primitivas no $EbRefPoint_ext$. Elas podem ser divididas em vários grupos, cada um implementado por meio de uma tecnologia diferente. Essa decisão de implementação depende da interface de serviços de estrutura da nuvem gerenciada. Por exemplo, algumas nuvens expõem seus serviços por Webservices Restful, enquanto outras as expõem por filas de mensagens. É por isso que a filosofia conveniente para essa interface é o padrão do

Função	Tarefas	Áreas Funcionais				
		F	C	A	P	S
Monitoring	Alert/Alarm	✓	✓	✓	✓	✓
	History	✓	✓	✓	✓	✓
	Alarm Correlation	✓	✓	✓	✓	✓
Accountability	Utilization		X	X	X	
	Accounting		✓	✓	✓	
Planning	Performance Analyses	✓	✓		✓	
	Simulation	✓	✓	✓	✓	✓
	Optimization	✓	✓	✓	✓	✓
	Dependability	✓	✓		✓	
Organizing	Discovery		✓	✓	✓	
	Resources Identity		✓	✓		✓
	Stockholders Identity		✓	✓		✓
Visualizing	History	✓				
	Utilization		✓	✓	✓	
	Alarms	✓	✓	✓		
	Dashboard	✓	✓	✓	✓	✓

Figura 6 – Áreas primárias da M-API

provedor de serviços. Na visualização $EbRefPoint_int$, a interface pode ser implementada por meio de construtores de comportamento (como interfaces na linguagem Java), enquanto a partir da visualização $EbRefPoint_ext$ ela pode ser implementada por um serviço Web

Restful desacoplando essa interface de sua tecnologia.

Apesar dessa discussão operacional, temos um grupo especial de serviços exposto por $EbRefPoint_ext$ que tem um papel fundamental na estrutura do SDMan. Este é o serviço de tarefas de gerenciamento ($EbRefPoint_ext.mts$) que possui primitivas por meio das quais os orquestradores e controladores de nuvem gerenciada podem solicitar tarefas de gerenciamento. Um exemplo disso é o serviço de monitoramento específico do objeto gerenciado ou uma tarefa básica de coleta de dados periódica. Usando este tipo de serviço, um controlador ou orquestrador de nuvem gerenciado não precisa coletar dados por si só. A vantagem dessa ideia é a redução potencial do uso de recursos de gerenciamento, já que vários "clientes"podem solicitar o mesmo serviço com o mesmo destino de dados. Assim, o SDMan configuraria a tarefa de coleta de dados apenas uma vez. O grupo de outras primitivas chave é aquele cujos destinos são a configuração da estrutura de nuvem gerenciada e os dados de gerenciamento interno ($EbRefPoint_ext.gcdata$). Este grupo é responsável pela importação de dados e mensagens específicos da nuvem gerenciada, tornando-os disponíveis para os subcomponentes ManApps e SDMan.

4.2.6 Modelo de Distribuição dos Componentes SDMan

A distribuição geográfica dos componentes do SDMan requer um estudo mais detalhado dos prós e contras de cada modelo, uma vez que cada um pode ser fragmentado, replicado ou uma combinação de ambos. Além disso, o modelo de distribuição precisa ser estudado com relação ao modelo de negócio *pay-as-you-go* das nuvens. Em um primeiro momento podemos sugerir dois modelos básicos: (i) O modelo replicado, que implica ter algumas cópias idênticas dos componentes, assim como seu banco de dados (Fig.7) em nuvens diferentes (por exemplo, replicar componentes no Google Cloud, Azzure e AWS). (ii) O modelo fragmentado propõe a disseminação estratégica dos componentes do SDMan em diferentes nuvens, bem como seus bancos de dados (Fig8). Para o modelo fragmentado, há algumas vantagens proeminentes na implantação dos bancos de dados distribuídos (DDB) (ÖZSU, 1999). Notavelmente:

- **Gestão transparente de dados distribuídos**: Alto nível de suporte oferecido para o desenvolvimento de aplicações complexas.
- **Autonomia Local**: Acesso mais rápido aos dados localizados próximos aos usuários, ao mesmo tempo, compartilhar dados armazenados em outro lugar. Também promove uma certa autonomia para o sistema, uma vez que os dados ganham independência.
- **Integridade e disponibilidade**: os bancos de dados distribuídos são projetados para melhorar a confiabilidade porque eles têm componentes replicados e, portanto, eliminam pontos únicos de falha. Se um banco de dados falhar ou um único link

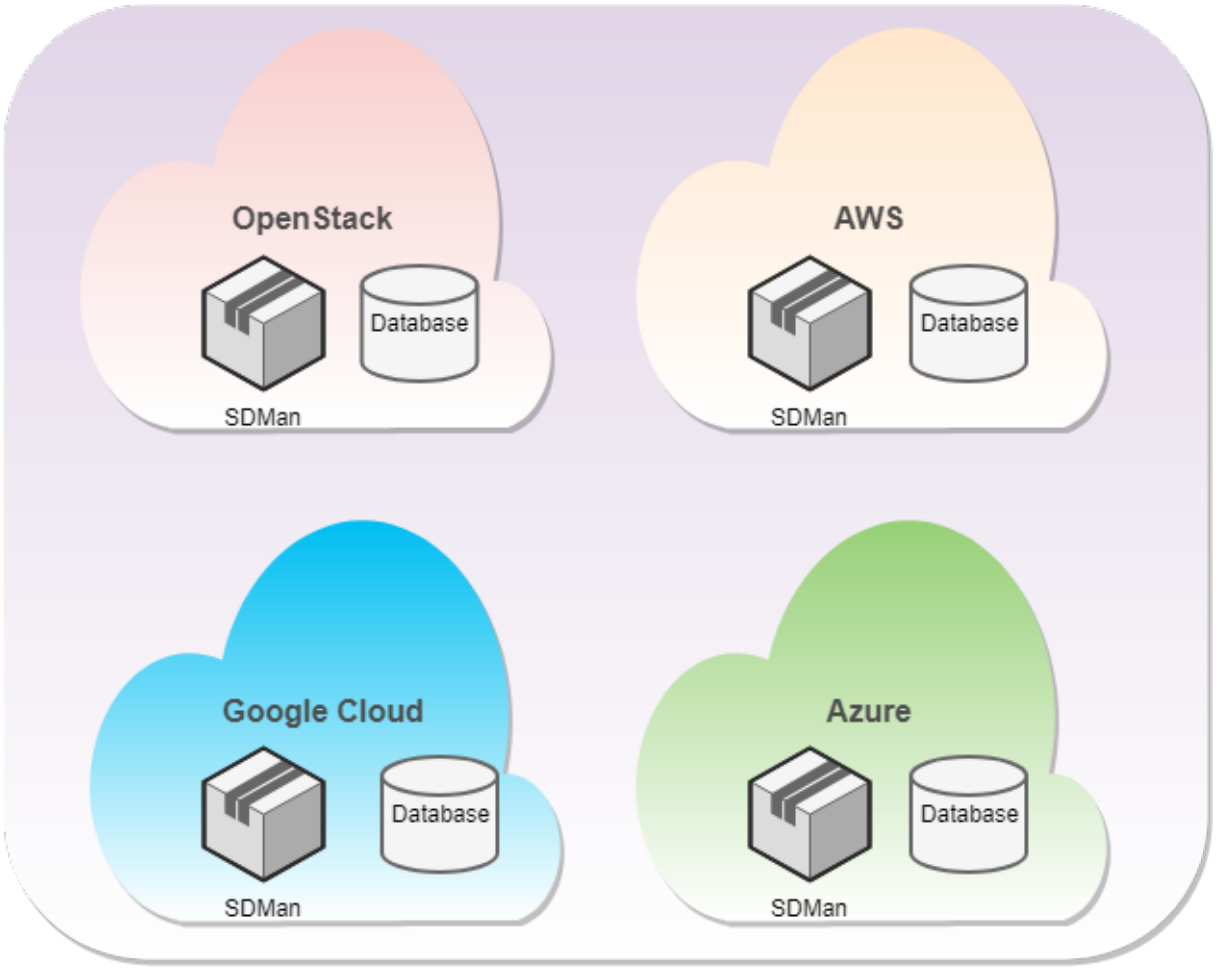

Figura 7 – Modelo SDMan com replicação

(caminho ou canal de comunicação entre dois componentes ou dispositivos) falhar ao tornar um ou mais bancos de dados inacessíveis, isso não é suficiente para deixar todo o sistema inativo.

- **Desempenho**: um DDB pode funcionar melhor do que um sistema centralizado por meio da distribuição de carga de trabalho. Como em cada nuvem haverá apenas parte do banco de dados, o esforço para os serviços da Unidade Central de Processamento (CPU) e serviços de E/S (entrada e saída de comunicações) é muito menor do que os esforços no caso de um banco de dados centralizado.

- **Custo de Escalonamento**: ao usar um DDB, os custos associados à expansão são menores quando comparados aos custos associados à expansão de um banco de dados centralizado.

- **Compartilhamento**: com vários bancos de dados distribuídos geograficamente e distantes entre si, a distribuição de dados é necessária, pois mantê-los centralizados pode

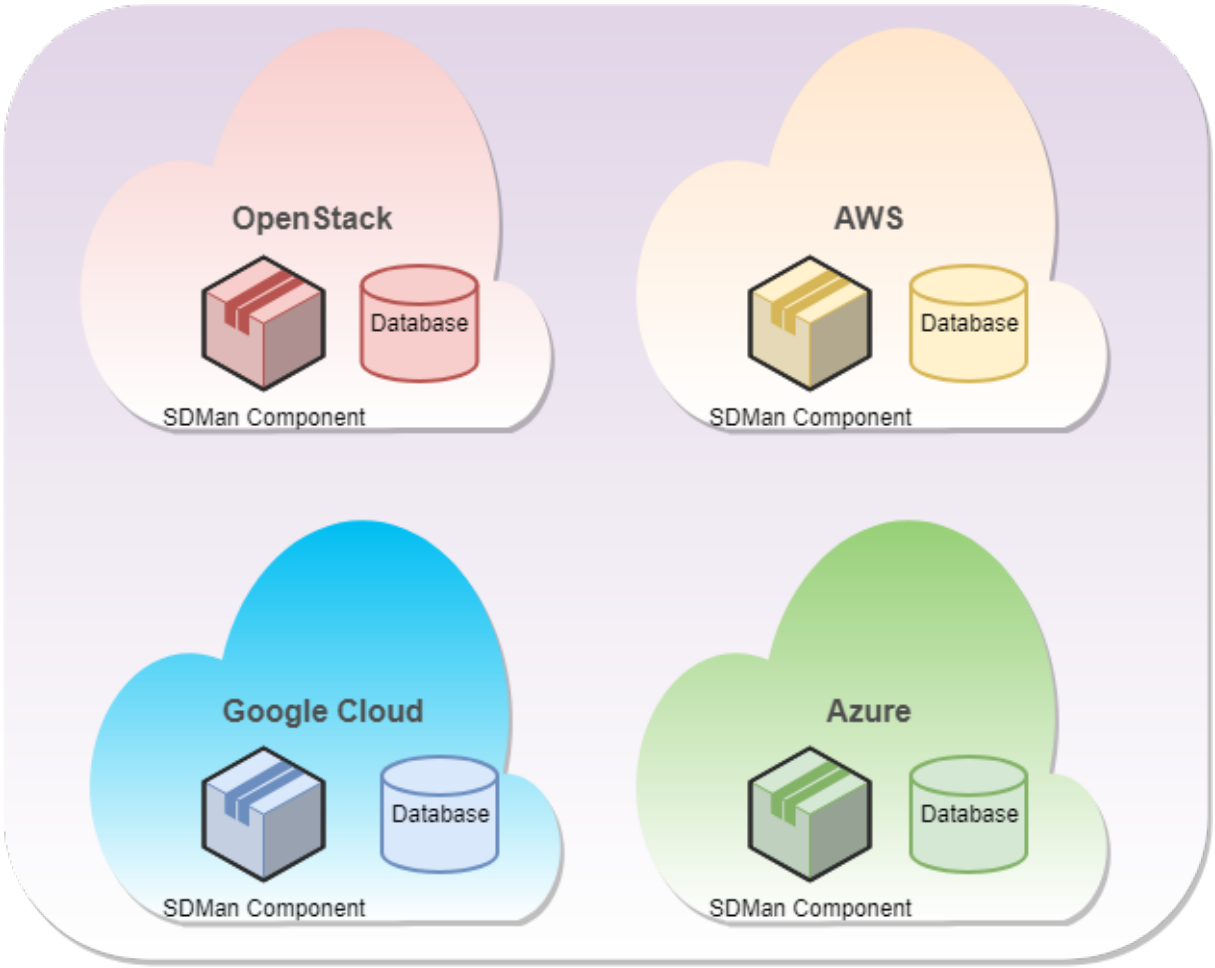

Figura 8 – Modelo SDMan fragmentado através das nuvens

envolver um alto custo. Além disso, os dados distribuídos podem ser compartilhados em várias regiões a um custo muito menor.

Infelizmente, há uma desvantagem importante e incisiva no uso de DDB, a complexidade. Além dos problemas encontrados em um ambiente de banco de dados centralizado, há vários problemas não resolvidos, como modelagem, processamento, consultas, simultaneidade, sistema operacional e assim por diante (ÖZSU, 1999).

4.2.7 SDMan em relação a perspectiva da computação de Fog e Edge

O Fog Computing é uma plataforma altamente virtualizada que fornece serviços de computação, armazenamento e rede entre dispositivos finais e data centers tradicionais de computação em nuvem, geralmente, mas não exclusivamente, nos limites da rede (AL-FUQAHA et al., 2015). Seguindo o conceito da Fog Computing, é plausível que o SDMan se ajuste ao cenário híbrido usando o modelo de distribuição fragmentada. Para Computação

de Borda pura (SHI et al., 2016), no entanto, o uso do SDMan não é trivial. Depende da capacidade, agrupamento e interconexão dos dispositivos de borda. Um grupo de dispositivos de borda funcionaria como uma nuvem fragmentada, suportando componentes SDMan.

4.2.8 SDMan Overhead

O SDMan foi projetado para ter uma estrutura de código leve e modular, usando Java em conjunto com a tecnologia OSGi Alliance (OSGI. . . , 2019) (anteriormente conhecida como a iniciativa Open Services Gateway). Os pacotes OSGI reduzem a complexidade do software, fornecendo uma arquitetura modular para sistemas distribuídos em larga escala. Portanto, esta subseção visa exibir o baixo consumo de recursos de hardware do SDMan.

Para tanto foi montado um cenário em que o controlador SDMan gerencia três hosts distintos, cada um contendo vários serviços, como mostrado na Fig.10, que recebem e enviam dados ao controlador. Ainda, por baixo dos panos, o SDMan está conectado a dois bancos de dados, o MongoDB, documental, e o banco de dados semântico Neo4j (Fig.10). Então, trocando informações continuamente com os hosts e seus serviços, é possível visualizar na Fig.9 como o consumo de recursos do SDMan se comporta. Notavelmente, após mais de oito horas de execução, baixo consumo de memória e processador, os números são suportados pela Fig.11. O controlador foi executado em um processador Intel i5-7300HQ.

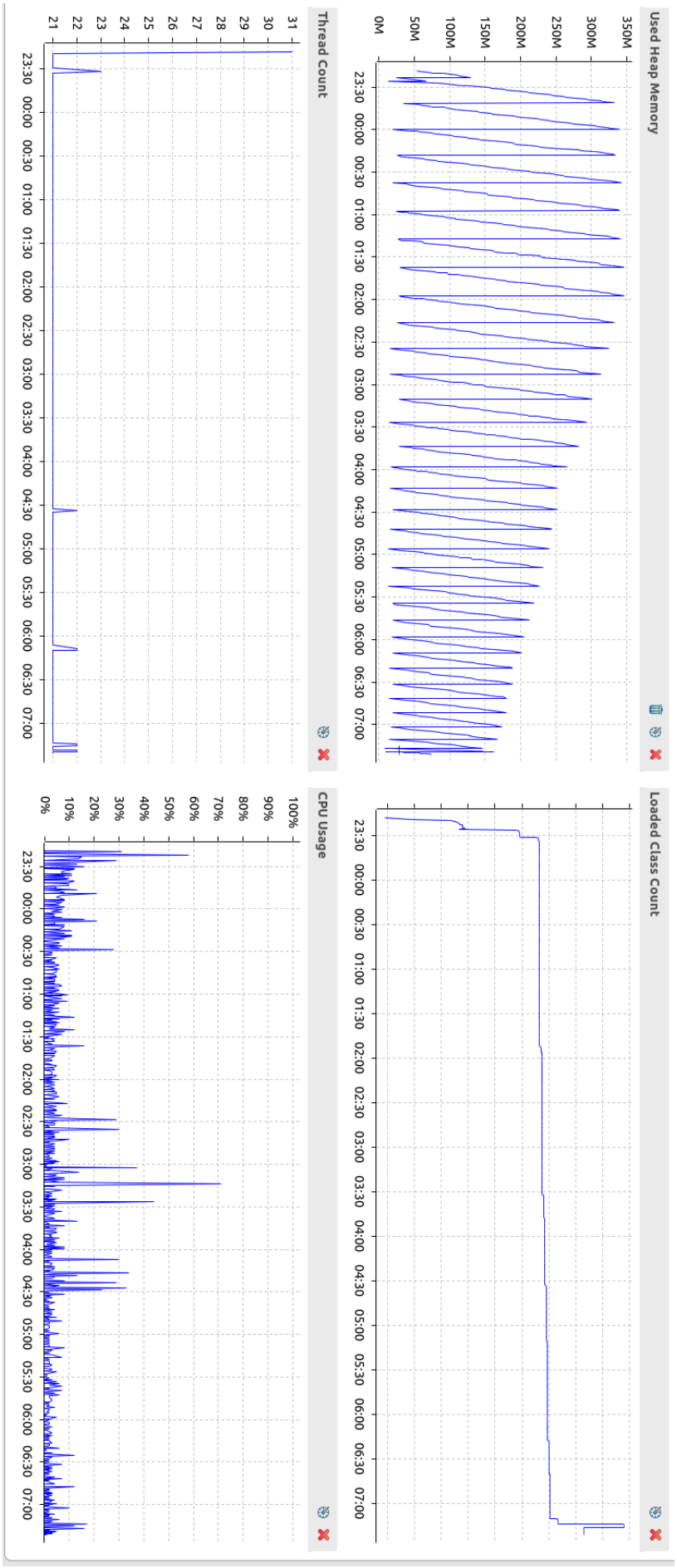

Figura 9 – SDMan - consumo de recursos

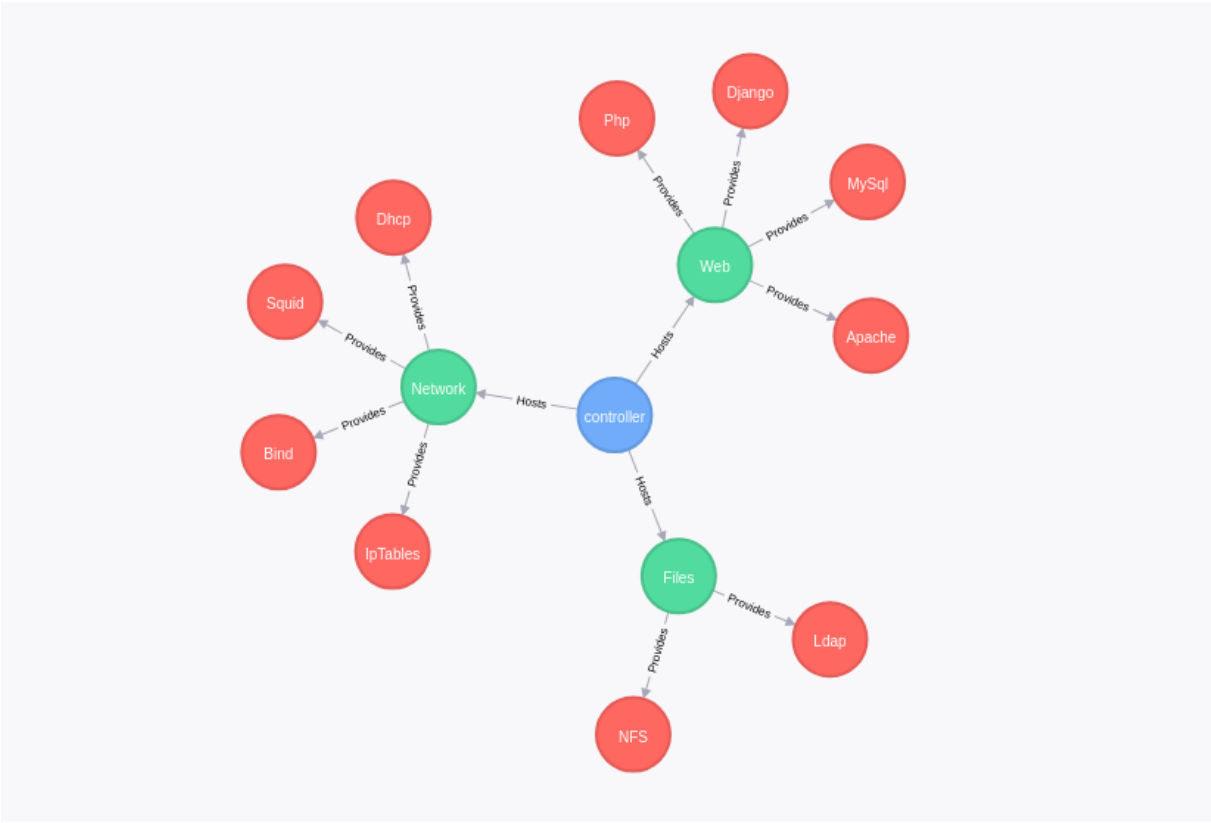

Figura 10 – Gráfico gerado através do Neo4J do controller SDMan e suas conexões

▼ Thread	
TotalStartedThreadCount	38
Thread count	21
Peak thread count	31
Daemon thread count	17
▼ Class loading	
TotalLoadedClassCount	4784
Loaded class count	4746

Figura 11 – SDMan - consumo de recursos 2

5 Implementação de referência do SDMan

Este capítulo introduz a implementação de referência do SDMan.

5.1 Implementação da interface Northbound (NbInt)

Nas Listagens 4.1 e 4.2, citadas em 4.2.4, são exibidas, a título de exemplo, a *Monitoring-History Abstract Factory class*, na notação da linguagem Java dentro do pacote Monitoring da classe concreta (CNagiosMonHistory ⇒ que estende MonHistory), e a *factory interface* (interface NagiosMonHistoryFactory) para o *NagiosVMF*, os métodos de *concrete factory* estão associados aos pacotes do VMF. Estes pacotes devem ser desenvolvidos pelo provedor VMF.

5.2 Implementação da Interface Eastbound (EbInt)

A interface Eastbound pode ser observada por dois pontos de referência diferentes. O primeiro é o ponto de referência interno, na fronteira entre o SDManCtrl e o Connector ($EbRefPoint_int$). O segundo é o ponto de referência externo, entre o Connector e a framework da nuvem gerenciada ($EbRefPoint_ext$) como mencionado em 4.2.5. Para estes dois pontos de referência, o conjunto de primitivas é o mesmo, mas a tecnologia de implementação pode variar, dependendo da tecnologia de nuvem de destino. As primitivas no $EbRefPoint_ext$ foram implementadas usando o Factory Design Pattern, ocultando os detalhes da tecnologia específica da nuvem de destino. Na visualização $EbRefPoint_int$, a interface foi implementada por meio de construtores de comportamento (interface na linguagem Java).

Os conectores, por sua vez, são adaptadores de software feitos sob medida para a tecnologia de nuvem de destino, o que significa que, para cada nuvem gerenciada (como o Google Cloud, AWS), é necessária uma implementação específica. Isso acontece porque não há um padrão definido de como cada uma dessas nuvens será capaz de fornecer informações sobre si mesma. O objetivo principal do Connector, manter uma M-API estável, mesmo se as APIs de gerenciamento de nuvem de destino divergirem. No nível de usuário do SDMan, existe apenas uma API (M-API), fornecendo transparência. Como resultado, foi utilizado o Factory Method, que permite a automação ao lidar com diversas nuvens, uma vez que um padrão genérico é definido na implementação. Ou seja, a finalidade do Factory Method é em várias classes que implementam a mesma operação, retornam o mesmo tipo abstrato, mas instanciam internamente diferentes classes que o implementam. Com o Factory Method, o criador do objeto faz uma escolha de qual classe instanciar para

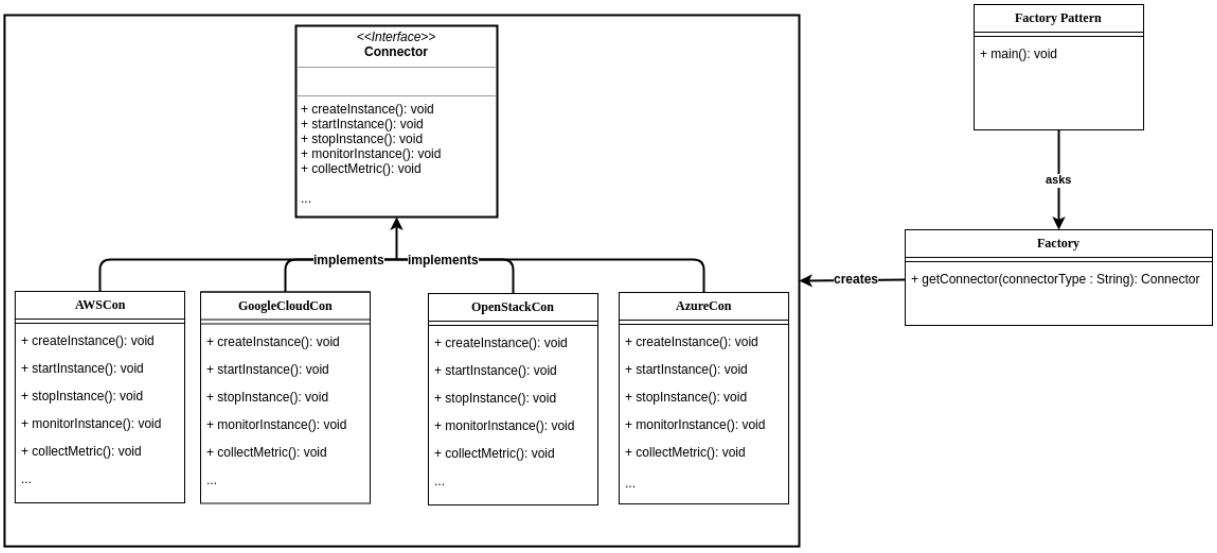

Figura 12 – Modelo do *Factory Method* para o Connector

o cliente. A Fig. 12 representa o modelo do *Factory Method*.

Os códigos a seguir mostram o uso do *Factory Method*, iniciando pela Listagem 5.1.

Listagem 5.1 – Interface Connector

```
1  package sdman.mapi.connector;
2
3  public interface Connector {
4
5      public abstract void createInstance();
6
7      public abstract void startInstance();
8
9      public abstract void stopInstance();
10
11     public abstract void monitorInstance();
12
13     public abstract void collectMetric();
14
15     ...
16
17 }
```

Em seguida, a implementação do conector para cada nuvem é feita, onde cada método da interface genérica do Connector deve ser substituído. Por exemplo, mostramos a implementação do conector AWS na listagem a Listagem 5.2.

Então, finalmente, a *Factory* pode ser de fato implementada a Listagem 5.3.

Assim, é possível manipular várias nuvens mesmo se o SDMan estiver em uso total, não é necessário interromper nada para lidar com uma ou outra nuvem. É o suficiente para o usuário escolher qual instância de qual nuvem ele quer e ele terá todas as informações possíveis que o SDMan fornece. Mesmo provisionando novos recursos em uma determinada nuvem ou liberando-os.

Listagem 5.2 – AWS Connector Connector

```
1  package sdman.mapi.connector;
2
3  public class AWSCon implements Connector {
4
5      @Override
6      public void createInstance() {
7
8      }
9
10     @Override
11     public void startInstance() {
12
13     }
14
15     @Override
16     public void stopInstance() {
17
18     }
19
20     @Override
21     public void monitorInstance() {
22
23     }
24
25     @Override
26     public void collectMetric() {
27
28     }
29
30     ...
31 }
```

Listagem 5.3 – Factory

```
1  package sdman.mapi.connector;
2
3  public class Factory {
4
5      public Connector getConnector (String connectorType) {
6
7          if (connectorType == null) {
8              return null;
9          }
10
11         if (connectorType.equalsIgnoreCase("AWS")) {
12             return new AWSCon();
13         } else if (connectorType.equalsIgnoreCase("GoogleCloud")) {
14             return new GoogleCloudCon();
15         } else if (connectorType.equalsIgnoreCase("OpenStack")) {
16             return new OpenStackCon();
17         } else if (connectorType.equalsIgnoreCase("Azure")) {
18             return new AzureCon();
19         }
20
21         return null;
22     }
23
24 }
```

Para ilustrar como a implementação de cada conector difere, devido à tecnologia adotada pela nuvem, no uso do *Factory Method*, foram listadas algumas das principais nuvens do mercado e a respectiva implementação do conector.

a) **Google Cloud**: oferece sua API de monitoramento, chamada Stackdriver, para extrair métricas de desempenho, tempo de atividade e integridade geral dos aplicativos dentro da tecnologia de nuvem (STACKDRIVER..., 2018). Que torna possível consultar e obter suas informações usando a API REST, conforme mostrado na Listagem 5.4.

Listagem 5.4 – Google Stackdriver

```
1 package sdman.connector.GoogleCloud
2
3 public void stackdriverConnector (String command) {
4 ...
5 }
```

Para definir quais métricas monitorar, essa função recebe como parâmetro uma String com um comando seguindo os padrões da API do StackDriver (CREATING..., 2018).

b) **AWS**: oferece uma solução um pouco mais sofisticada com o AWS SDK para Java, que fornece uma API Java para o Amazon Web Services. Com o SDK, é possível construir aplicativos Java que se integram ao Amazon EC2, Amazon S3 e Amazon SimpleDB (e mais) (AWS..., 2018), o que torna possível extrair várias métricas, incluindo métricas de desempenho, como exemplificado pelo seguintes códigos.

A Listagem 5.5 mostra como criar uma instância no Amazon EC2:

Listagem 5.5 – AWS create instance

```
1 package sdman.connector.AWS;
2
3 public class CreateInstance
4 {
5     public void createInstance(String name, String ami_id, String instanceType,
          String accessKey, String secretKey, String region)
6     {
7         ...
8     }
9 }
```

Após a instância ser criada, é necessário iniciar o serviço de monitoramento para ela, que pode ser feito como na Listagem 5.6.

Com o monitoramento ativo, é possível coletar algumas métricas como CPUUtilization, NetworkIn, NetworkOut, DiskReadOps, DiskWriteOps, DiskReadBytes, DiskWriteBytes (AMAZON..., 2018), exemplificado pela Listagem 5.7.

Listagem 5.6 – AWS Monitor

```
1 public void monitorInstance(String instance_id, String accessKey, String
      secretKey, String region) {
2         ...
3 }
```

Listagem 5.7 – AWS Find Cloud Watch

```
1 public void findCloudWatchData(String instance_id, String accessKey, String
      secretKey, String region, String metricName, long offsetInMilliseconds, int
      period) {
2     ...
3  }
```

E assim, finalmente as métricas são coletadas de uma instância da AWS para SDMan.

c) **OpenStack**: como é um software de código aberto, você tem acesso aos seus módulos (como o Nova, Keystone, Glance, Neutron, Horizon, Ceilometer e outros) (OPENSTACK, 2019) e também para o hardware do datacenter. No entanto, nosso foco para o Connector é o Ceilometer, que é um serviço de coleta de dados que oferece a capacidade de medir métricas e eventos de desempenho em todos os componentes do OpenStack. Para armazenamento e recuperação de dados, foi utilizado um serviço de banco de dados temporal baseado em nuvem, o Gnocchi. O Gnocchi é um banco de dados de código aberto que permite o armazenamento e a indexação de dados e recursos de séries temporais em grande escala. Então, por meio do Gnocchi, é possível coletar métricas de desempenho geradas pelo Ceilometer (SANTOS JOAO H. G. MEDEIROS, 2018). Abaixo é mostrado, na Listagem 5.8 o código central, usando a API gnocchiclient (GNOCCHI,), em python.

Listagem 5.8 – Openstack Connector

```
1 import json
2 import shade
3 import os
4
5 from keystoneauth1.identity import v3
6 from keystoneauth1 import session
7 from gnocchiclient.v1 import client
8
9 class Gnocchi():
10     def __init__(self):
11
12     def get_metric_memory_usage(self,resource_id,start,stop,granularity):
13
14     def get_metric_cpu_utilization(self,resource_id,start,stop,granularity):
15
16     def get_list_meters(self,resource_id):
17
18     def get_list_resources(self):
```

d) **Microsoft Azure**: Embora a implementação do conector do Azure ainda não tenha sido concluída, a Microsoft fornece meios usando a API REST para que as métricas da nuvem sejam coletadas programaticamente por meio de bibliotecas cliente que fornecem uma interface nativa (.NET, Java, Node.js, Python e Azure CLI 2.0 SDK) (AZURE...,) (OVERVIEW...,), possibilitando o acesso a todas as métricas no período de tempo desejado.

5.3 Implementação do Repositório de Persistência

Como mencionado anteriormente na seção 2.2.6, no caso de usar um único banco de dados centralizado onde todas as operações são executadas no único servidor de banco de dados, com o tempo, é muito provável que o fluxo de tráfego aumente para que as operações no sistema sejam lentas pelo tamanho do banco de dados ou pelo número de solicitações ou junções ruins, além da possibilidade de o banco de dados estar inativo, tornando o aplicativo inteiro indisponível também.

Uma maneira de resolver isso é desfazendo a dependência no banco de dados único. Isso pode ser feito dividindo os dados de acordo com alguns critérios convenientes e colocando-os em hosts separados, por exemplo, colocando os dados de leitura pesados em um host e de gravação leve em outro. De qualquer forma, a implantação de algo como isto irá redesenhar o banco de dados e como ele está disposto e onde quer que uma nova estratégia de tipo seja implementada, levará muito mais tempo e trabalho para implantá-lo.

A estratégia pretendida inclui adotar um proxy para resolver este problema de uma maneira menos dolorosa e muito mais elegante. Por meio do ProxySQL (ProxySQL,), é possível fazer isso com pouca mudança de código, além de ajudar nos cenários de desempenho e segurança. ProxySQL é uma ferramenta de código aberto escrita em C++ e parte dela é escrita em Python. É um serviço de proxy que adquire solicitações MYSQL de clientes e, em seguida, os transmite para os serviços de back-end e, finalmente, encaminha o resultado recebido para o solicitante. Ele permite rotear consultas SQL para vários bancos de dados baseados em regras com a capacidade de monitorar consultas, desempenho e estatísticas úteis para otimizar o aplicativo.

Na Fig. 13 Um exemplo de uso é mostrado onde há dois aplicativos diferentes se conectando ao ProxySQL. ProxySQL encaminha consultas para hosts com base em regras. O banco de dados semântico armazena as relações entre os VMFs e suas bases operacionais.

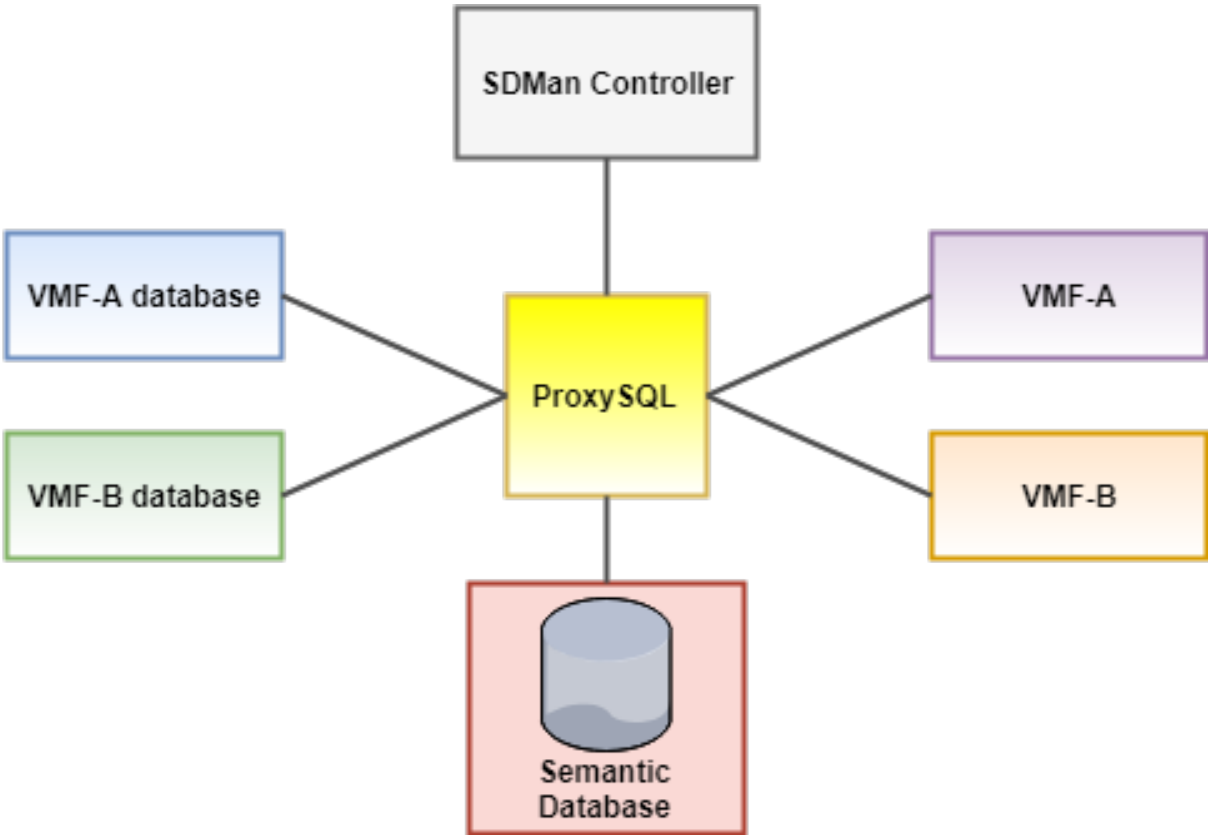

Figura 13 – VMFs fazendo uso do ProxySQL

5.4 Integração do SDMan com plataformas de nuvem

Esta seção tem como objetivo demonstrar a possibilidade de integração com diversas plataformas de nuvem através de chamadas de código desenvolvidas para o Connector do SDMan, passandos pelas nuvens do Google Cloud, AWS, Azure e também pelo OpenStack. Os código citados nesta seção podem ser encontrados em (SDMANCONN, 2019)

a) **Google Cloud**: Por exemplo, para extrair a utilização da CPU de uma instância específica e em um determinado período, a consulta é exibida na Listagem 5.9.

Listagem 5.9 – Stackdriver query

```
1 "https://monitoring.googleapis.com/v3/\\projects/sdman-196518/timeSeries/?filter
    =\\metric.type+\%3D+\%22compute.googleapis\\.com\%2Finstance\%2Fcpu\%2
    Futilization\%2\\2\&interval.endTime=2018-08-09T00\%3A00\%3A00.000000Z\&
    interval.start\\Time=2018-08-08T00\%3A00\%3A00.000000Z"
```

A consulta citada acima resulta no uso da CPU da instância em um período de 24 horas (em intervalos de 60 segundos), entre 2018-08-08 e 2018-08-09, como exibido na Listagem 5.10.

Embora tenhamos acesso a essas informações graficamente por meio do console do Google Cloud (Fig. 14), isso não nos fornece uma solução inteligente ou automatizada.

Listagem 5.10 – Stackdriver response

```
{
  "timeSeries": [
    {
      ...
      "metricKind": "GAUGE",
      "valueType": "DOUBLE",
      "points": [
        {
          "interval": {
            "startTime": "2018-08-08T17:00:42.583Z",
            "endTime": "2018-08-08T17:00:42.583Z"
          },
          "value": {
            "doubleValue": 1.0387515795042557
          }
        },
        ...
      ]
    }
  ]
}
```

A maneira proposta de se fazer isso é coletar métricas e armazená-las nos bancos de dados do SDMan.

b) **AWS**: Como exemplo, na Listagem 5.11, a utilização da CPU foi escolhida para demonstrar a viabilidade, passando os parâmetros corretos para o método "findCloudWatchData"(para a demonstração, foi escolhido coletar informações das últimas 24 horas a cada intervalo de um minuto, de 2018-08-08).

Listagem 5.11 – AWS response

```
Wed Aug 08 14:53:00 BRT 2018 / 0.0 / Percent
Wed Aug 08 14:54:00 BRT 2018 / 0.166666666666667 / Percent
Wed Aug 08 14:55:00 BRT 2018 / 0.0 / Percent
...
```

Assim como o Google Cloud, a AWS disponibiliza essas informações no seu painel do EC2 Fig. 15.

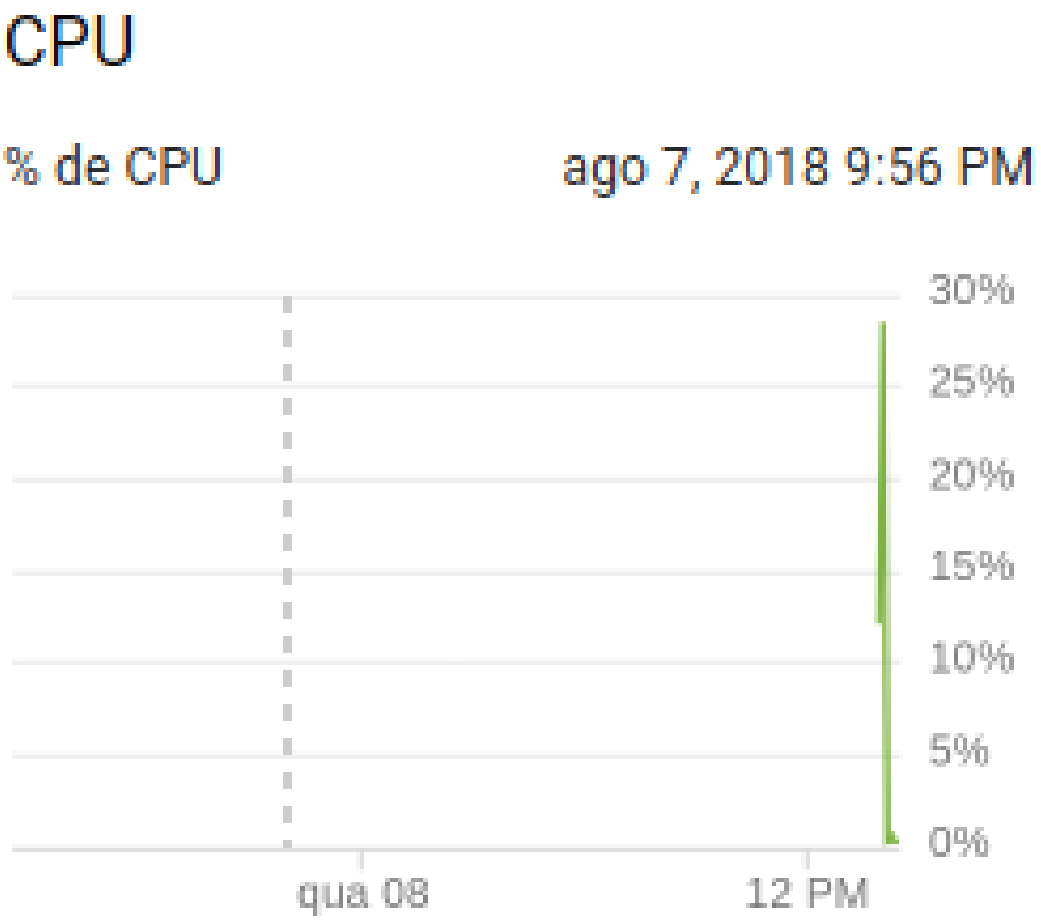

Figura 14 – Google Cloud: Utilização da CPU

c) **OpenStack**: Monitoramento do uso da RAM, exibido na Listagem 5.12, de uma instância em um curto período, demonstrando o código em execução.

Listagem 5.12 – OpenStack response

```
vm1
flavor: m1.large
2VCPUs 4GB RAM 20GB HD
System:Ubuntu 16.04

|timestamp|granularity|value(MB)|

|2018-08-09 14:08:18|1|55.0|
|2018-08-09 14:08:19|1|55.0|
|2018-08-09 14:08:20|1|134.0|
...
```

O OpenStack não possui uma ferramenta de monitoramento gráfico nativa em sua instalação padrão. Neste caso, o Grafana foi integrado. Assim, pode-se visualizar os dados sendo gerados e até mesmo verificar se as métricas estão sendo coletadas corretamente como na Fig. 16.

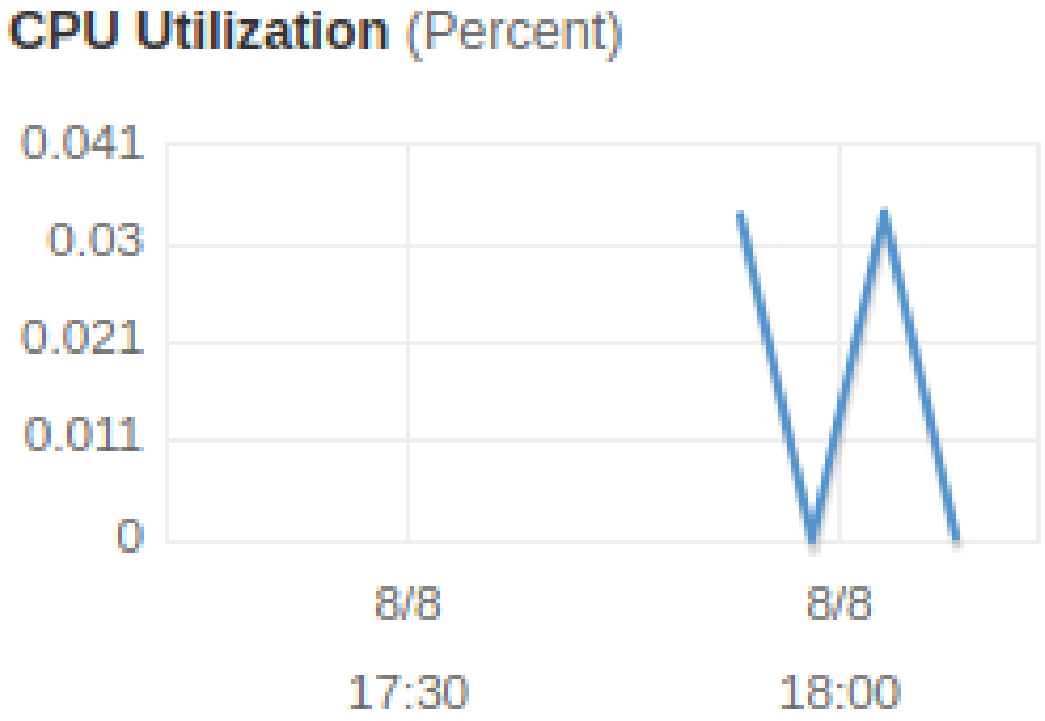

Figura 15 – AWS EC2: Utilização da CPU

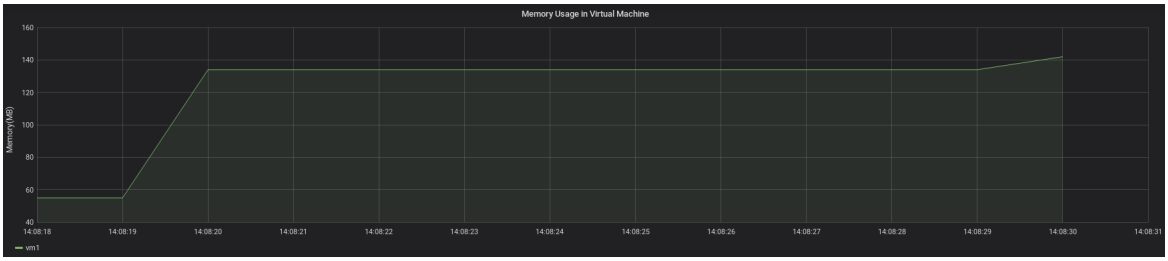

Figura 16 – OpenStack: uso de RAM provido pelo Grafana

6 Conclusão

Este capítulo visa apresentar os resultados da proposição do SDMan no âmbito acadêmico e científico, bem como as considerações finais e trabalhos futuros.

6.1 Considerações Finais

Este trabalho apresenta a implementação de referência do SDMan, um passo à frente em uma estrutura de gerenciamento definida por software, dentro do contexto de computação em Nuvem.

As ferramentas de gerenciamento de plataformas de nuvem disponíveis no mercado foram concebidas anteriormente ao início da era da computação em nuvem e foram adaptadas para suprir a nova demanda. Como resultado estas soluções, até o momento, não são capazes de suprir pontos necessários da gestão da SDI citados em 2.4.

O SDMan foi concebido de forma a possuir os principais recursos de uma solução verdadeiramente definida por software, como a capacidade de aceitar conjuntos de instruções que podem alterar o comportamento da infraestrutura de nuvem gerenciada e o controle separado da execução. O SDMan é capaz de sanar as lacunas levantadas na seção 2.4 e reforça a integração como estratégia chave para os serviços de nuvem de nova geração, destacando a reutilização de ferramentas de gerenciamento tradicionais como Virtual Management Functions (VMF). A implementação de referência revela as principais estratégias por trás dos componentes do SDMan.

6.2 Trabalhos Futuros

Notadamente um dos grandes desafios para o SDMan é lidar com toda a complexidade de um sistema fragmentado ao longo das plataformas de Nuvem. Não é um trabalho simples e carece que esteja em constante aprimoramento. É preciso salientar que até mesmo diferentes disposições geográficas podem influenciar sobre qual modelo de distribuição é o mais adequado, bem como custo e complexidade.

No entanto o trabalho futuro de maior importância é sair do âmbito da prototipagem e torná-lo um produto capaz de, primeiramente ser implantado e posteriormente testado, para que seja possível observá-lo em situações reais para analisar de forma detalhada seu comportamento de consumo de recursos e também sua eficiência.

Outro ponto importante a ser observado é a utilização das tecnologias adotadas no Framework, em especial o Cluster Control, que embora seja um grande aliado para

replicar bancos de dados distribuídos e também capaz de utilizar o ProxySQL, é preciso avaliar o nível de dependência que a adoção desta tecnologia pode causar.

Referências

AL-FUQAHA, A. et al. Internet of things: A survey on enabling technologies, protocols, and applications. *IEEE communications surveys & tutorials*, IEEE, v. 17, n. 4, p. 2347–2376, 2015. Citado na página 46.

ALASHOOR, T. Cloud computing: a review of security issues and solutions. *International Journal of Cloud Computing*, v. 3, n. 3, p. 228–244, 2014. Citado na página 17.

ALLIANCE, O. *Osgi service platform, release 3*. [S.l.]: IOS Press, Inc., 2003. Citado na página 28.

AMAZON. *Amazon CloudWatch Documentation*. [S.l.], 2017. Disponível em: <https://aws.amazon.com/pt/documentation/cloudwatch/>. Citado na página 33.

AMAZON Elastic Compute Cloud. 2018. https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/monitoring_ec2.html. Acessado em: 2018-07-27. Citado na página 54.

AWS SDK for Java. 2018. https://docs.aws.amazon.com/sdk-for-java/v1/developer-guide/welcome.html. Acessado em: 2018-07-27. Citado na página 54.

AZURE REST API Reference. https://docs.microsoft.com/pt-br/rest/api/azure/. Acessado em: 2018-07-27. Citado na página 56.

BLACK, U. D. *Network Management Standards: SNMP, CMIP, TMN, MIBs and Object Libraries*. [S.l.]: McGraw-Hill, Inc., 1994. Citado na página 39.

CHAPPELL, C. Nfv mano: What's wrong & how to fix it. *Heavy Reading*, v. 13, n. 2, 2015. Citado na página 24.

CHERRIER, S. et al. Salt: a simple application logic description using transducers for internet of things. In: IEEE. *Communications (ICC), 2013 IEEE International Conference on*. [S.l.], 2013. p. 3006–3011. Citado na página 42.

CLOUDMONIX. *Cloudmonix - Advanced Cloud Monitoring and Automation Tools*. [S.l.], 2015. Disponível em: <http://www.cloudmonix.com/>. Citado na página 33.

CLOUDSTACK, A. *Apache CloudStack^{TM} - Open Source Cloud Computing^{TM}*. [S.l.], 2017. Disponível em: <https://cloudstack.apache.org/>. Citado na página 26.

CLUSTERCONTROL. 2019. https://severalnines.com/product/clustercontrol. Acessado em: 2019-03-22. Citado na página 30.

CREATING Custom Metrics. 2018. https://cloud.google.com/monitoring/custom-metrics/creating-metrics. Acessado em: 2018-07-27. Citado na página 54.

ETSI-MANO. 2017. http://www.etsi.org/technologies-clusters/technologies/nfv/open-source-mano. Acessado em: 2019-02-12. Citado 2 vezes nas páginas 18 e 24.

FATEMA, K. et al. A survey of cloud monitoring tools: Taxonomy, capabilities and objectives. *Journal of Parallel and Distributed Computing*, v. 74, n. 10, p. 2918 – 2933, 2014. ISSN 0743-7315. Citado na página 33.

GNOCCHI. *Gnocchi Client API*. https://gnocchi.xyz/gnocchiclient/api.html. Acessado em: 2018-07-27. Citado na página 55.

HAN, B. et al. Network function virtualization: Challenges and opportunities for innovations. *IEEE Communications Magazine*, IEEE, v. 53, n. 2, p. 90–97, 2015. Citado na página 23.

HAN, B. et al. Network function virtualization: Challenges and opportunities for innovations. *IEEE Communications Magazine*, IEEE, v. 53, n. 2, p. 90–97, 2015. Citado na página 24.

JARARWEH, Y. et al. Software defined cloud: Survey, system and evaluation. *Future Generation Computer Systems*, Elsevier, v. 58, p. 56–74, 2016. Citado na página 17.

KANDIRAJU, G. et al. Software defined infrastructures. *IBM Journal of Research and Development*, IBM, v. 58, n. 2/3, p. 2–1, 2014. Citado 2 vezes nas páginas 17 e 23.

MELL, P.; GRANCE, T. *The NIST Definition of Cloud Computing*. Gaithersburg, MD, 2011. Citado 2 vezes nas páginas 17 e 21.

MIJUMBI, R. et al. Network function virtualization: State-of-the-art and research challenges. *IEEE Communications Surveys & Tutorials*, IEEE, v. 18, n. 1, p. 236–262, 2016. Citado 2 vezes nas páginas 18 e 24.

MONITIS. *Network and IT Monitoring Systems*. [S.l.], 2017. Disponível em: <http://www.monitis.com/>. Citado na página 33.

MONTEIRO, M. E. et al. An ontology-based approach to improve snmp support for autonomic management. In: IEEE. *Network and Service Management (CNSM), 2014 10th International Conference on*. [S.l.], 2014. p. 280–283. Citado na página 37.

MORENO-VOZMEDIANO, R.; MONTERO, R. S.; LLORENTE, I. M. Key challenges in cloud computing: Enabling the future internet of services. *IEEE Internet Computing*, IEEE, v. 17, n. 4, p. 18–25, 2013. Citado na página 24.

NAGIOS. *The Industry Standard In IT Infrastructure Monitoring*. [S.l.], 2017. Disponível em: <https://www.nagios.org/>. Citado na página 33.

OPENNEBULA.ORG. *OpenNebula: Flexible Enterprise Cloud Made Simple*. [S.l.], 2017. Disponível em: <https://opennebula.org/>. Citado na página 33.

OPENSTACK. *OpenStack*. 2019. https://www.openstack.org/. Acessado em: 2019-07-27. Citado 2 vezes nas páginas 17 e 55.

OPENSTACKFUEL. 2017. https://wiki.openstack.org/wiki/Fuel_CLI. Acessado em: 2017-04-23. Citado na página 33.

OSGI Alliance. 2019. https://web.archive.org/web/20120628185306/http://www.osgi.org/Technology/HomePage. Acessado em: 2018-07-27. Citado na página 47.

OVERVIEW of Metrics in Microsoft Azure. https://docs.microsoft.com/en-us/azure/monitoring-and-diagnostics/insights-how-to-customize-monitoring. Acessado em: 2018-07-27. Citado na página 56.

ProxySQL. https://proxysql.com. Acessado em: 2018-10-01. Citado 2 vezes nas páginas 28 e 56.

PUPPET. https://puppet.com/. Acessado em: 2017-04-23. Citado na página 42.

RACKSPACE. *Custom Infrastructure Monitoring*. [S.l.], 2017. Disponível em: <https://www.rackspace.com/cloud/monitoring>. Citado na página 33.

Rightscale. https://www.rightscale.com/learn/cloud- operations/cloud-monitoring-automation. Acessado em: 2017-04-23. Citado na página 33.

SANTOS JOAO H. G. MEDEIROS, R. P. F. M. d. Q. D. G. C. A. P. d. C. R. F. V. M. R. N. R. R. S. V. P. B. Monitoramento de recursos para aplicações de robótica em espaços inteligentes baseados em uma nuvem openstack. *XVI Workshop em Clouds e Aplicações (WCGA)*, may 2018. Citado na página 55.

SATYANARAYANAN, M. et al. The case for vm-based cloudlets in mobile computing. *IEEE pervasive Computing*, IEEE, n. 4, p. 14–23, 2009. Citado na página 17.

SDMANCONN. 2019. https://gitlab.com/ksimonassi/sdmanconn. Citado na página 57.

SEFRAOUI, O.; AISSAOUI, M.; ELEULDJ, M. Openstack: toward an open-source solution for cloud computing. *International Journal of Computer Applications*, Foundation of Computer Science, v. 55, n. 3, 2012. Citado na página 26.

SHI, W. et al. Edge computing: Vision and challenges. *IEEE Internet of Things Journal*, IEEE, v. 3, n. 5, p. 637–646, 2016. Citado 2 vezes nas páginas 17 e 47.

SOFTWARE-DEFINED Network Management: What We Need. 2017. http://www.networkcomputing.com/unified-communications/software-defined-network-management-what-we-need/149846456 . Acessado em: 2017-04-23. Citado 2 vezes nas páginas 18 e 24.

SON, J.; BUYYA, R. Sdcon: Integrated control platform for software-defined clouds. *IEEE Transactions on Parallel and Distributed Systems*, IEEE, v. 30, n. 1, p. 230–244, 2019. Citado na página 33.

STACKDRIVER Monitoring Documentation. 2018. https://cloud.google.com/monitoring/docs/. Acessado em: 2018-07-27. Citado na página 54.

TECHNOLOGIES, C. *CA Unified Infrastructure Management (CA UIM)*. [S.l.], 2017. Disponível em: <https://www.ca.com/us/products/ca-unified-infrastructure-management.html>. Citado na página 33.

THAMES, L.; SCHAEFER, D. Software-defined cloud manufacturing for industry 4.0. *Procedia CIRP*, Elsevier, v. 52, p. 12–17, 2016. Citado na página 33.

VAQUERO, L. M.; RODERO-MERINO, L. Finding your way in the fog: Towards a comprehensive definition of fog computing. *ACM SIGCOMM Computer Communication Review*, ACM, v. 44, n. 5, p. 27–32, 2014. Citado na página 17.

WICKBOLDT, J. A. et al. Software-defined networking: management requirements and challenges. *IEEE Communications Magazine*, IEEE, v. 53, n. 1, p. 278–285, 2015. Citado 2 vezes nas páginas 18 e 24.

ÖZSU, P. V. M. T. *Principles of distributed database systems*. [S.l.]: Prentice-Hall, 1999. Citado 2 vezes nas páginas 44 e 46.