UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO
CENTRO TECNOLÓGICO
DEPARTAMENTO DE INFORMÁTICA

RENAN FREIRE TAVARES

UM MIDDLEWARE PARA ORQUESTRAÇÃO DA GERÊNCIA DE RECURSOS E SERVIÇOS EM NUVEM

VITÓRIA
2016

RENAN FREIRE TAVARES

UM MIDDLEWARE PARA ORQUESTRAÇÃO DA GERÊNCIA DE RECURSOS E SERVIÇOS EM NUVEM

Monografia apresentada à Universidade Federal do Espírito Santo como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

Orientador: Prof. Dr. Magnos Martinello

VITÓRIA

2016

RENAN FREIRE TAVARES

UM MIDDLEWARE PARA ORQUESTRAÇÃO DA GERÊNCIA DE RECURSOS E SERVIÇOS EM NUVEM

COMISSÃO EXAMINADORA

_____

Prof. Dr. Magnos Martinello
Departamento de Informática – UFES
Orientador

_____

Prof. Dr. Maxwell Eduardo Monteiro
Instituto Federal do Espírito Santo

_____

Profª. M.Sc. Cristina Klippel Dominicini

Vitória, 21 de julho de 2016.

Agradeço em primeiro lugar aos meus pais que me possibilitaram chegar aqui.

Aos meus amigos e familiares que deram o suporte através de infinitas maneiras.

Ao Núcleo de Cidadania Digital por ceder parte da infraestrutura para os primeiros testes.

Ao professor Max pela dedicação e paciência na orientação deste trabalho.

Aos membros da banca por toda contribuição dada à este trabalho.

RESUMO

A computação em nuvem oferece uma infinidade de benefícios, principalmente em termos de escalabilidade e flexibilidade. Todavia, dado o elevado número de recursos quando comparado ao modelo tradicional, um ambiente em nuvem possui ferramentas capturando e produzindo informação de forma independente. Como resultado, os sistemas podem tomar conclusões imprecisas. Desse modo, os ambientes em nuvem necessitam que suas aplicações atuem de forma integrada. No que tange às aplicações de monitoramento, as mesmas precisam representar a topologia de recursos, analisando a dependência entre os dispositivos físicos, virtuais e suas respectivas aplicações. Por outro lado, cada uma realiza inferência sobre o item monitorado sem ter uma noção do todo bem estabelecida. Desse modo, o monitoramento pode se tornar inconsistente. Este trabalho apresenta uma proposta de orquestração da gerência de recursos em um ambiente em nuvem. As informações coletadas da infraestrutura são disponibilizadas através de um grafo semântico. Assim, abre-se a possibilidade para que seja feito o monitoramento sob demanda, escolhendo: a aplicação, os recursos e os dados a gerenciar.

Palavras-chave: Computação em Nuvem, Monitoramento em Nuvem, Orquestração de Recursos.

LISTA DE FIGURAS

LISTA DE SIGLAS

XaaS - *Anything as a Service*

SaaS - *Software as a service*

PaaS - *Plataform as a Service*

IaaS - *Infrastructure as a Service*

API - *Application Programming Interface*

SDN - *Software Defined Networking*

SGBD - Sistema Gerenciador de Banco de Dados

DNS - *Domain Name System*

TCP - *Transmission Control Protocol*

IP - *Internet Protocol*

ICMP - *Internet Control Message Protocol*

UDP - *User Datagram Protocol*

XML - *Extensible Markup Language*

SQL - *Structured Query Language*

JDBC - *Java Database Connectivity*

REST - *Representational State Transfer*

URL - *Uniform Resource Locator*

HTTP - *Hyper Text Transfer Protocol*

HTTPS - *Hyper Text Transfer Protocol Secure*

UUID - *Universally Unique Identifier*

SSH - *Secure Shell*

NRPE - *Nagios Remote Plugin Executor*

VM - *Virtual Machine*

MAC - *Media Access Control*

RAM - *Random Access Memory*

SUMÁRIO

1 INTRODUÇÃO

Nos últimos dez anos, a computação em nuvem se tornou um tema bastante discutido. A maioria dos serviços em nuvem fornece uma interface centralizada onde os recursos podem ser providos sob demanda. Desse modo, o ambiente virtual funciona de forma transparente e dinâmica para os usuários, sendo capaz de orquestrar recursos de diferentes tipos e fontes. Por exemplo, pode-se personalizar configurações de software, performance de hardware e características de rede. Portanto, as principais características da nuvem são escalabilidade e elasticidade, visto que plataformas em nuvem devem ser flexíveis para atender as exigências de um número potencialmente grande de usuários (WANG et al., 2010).

Em virtude deste número maior de recursos quando comparado ao modelo tradicional, um ambiente em nuvem utiliza-se da virtualização de recursos e necessita de sistemas de monitoramento mais complexos, escaláveis, robustos e ágeis. Portanto, sistemas de monitoramento devem ser refinados e adaptados para diferentes situações em ambientes de dinâmicos como as nuvens (ACETO et al., 2013).

Em (ACETO et al., 2013), são mencionadas algumas questões inerentes ao monitoramento em nuvem, são elas:

- Escalabilidade - A aplicação deve ser capaz de coletar, transferir e analisar uma grande quantidade de dados de forma eficiente.
- Elasticidade - A ferramenta deve suportar o aumento bem como a redução do conjunto de recursos monitorados.
- Adaptabilidade - O sistema deve evitar, sempre que possível, impactar, através do monitoramento, o desempenho de tarefas comuns da nuvem.
- Conveniência - A aplicação deve detectar eventos no momento exato (WANG et al., 2011).
- Autonomicidade - A ferramenta deve ser capaz de autogerenciar seus recursos, reagindo automaticamente a mudanças imprevisíveis, mantendo provedor e consumidor distantes de sua complexidade (MIAN; MARTIN; VAZQUEZ-POLETTI, 2013).

- Abrangência, extensibilidade e intromissão - O sistema deve: suportar diferentes tipos de recursos, diferentes tipos de dados de monitoramento e múltiplos locatários (HASSELMEYER; D'HEUREUSE, 2010); ser facilmente estendido através de *plugins* e módulos; e determinar quase nenhuma modificação na nuvem (KATSAROS; KÜBERT; GALLIZO, 2011).
- Resiliência, confiabilidade e disponibilidade - A aplicação deve: resistir a falhas enquanto continua operando normalmente, executar funções sob condições estabelecidas e prover os serviços projetados sempre que houver uma requisição do usuário (SHIREY, 2007). Com a virtualização, os recursos monitorados podem migrar de uma máquina física para outra, invalidando a lógica clássica de monitoramento e prejudicando a confiabilidade da aplicação, por exemplo.
- Precisão - O sistema deve prover medidas precisas, pois impactam diretamente na execução dos recursos. Em um ambiente virtual, coletar dados imprecisos pode resultar em perda de capital para os fornecedores, caso acordos a nível de serviço sejam quebrados.

Há um longo caminho para que o monitoramento de recursos em nuvem alcance maturidade. Isso porque, em um ambiente onde aplicações geram um grande volume de dados, a requisição e o acesso à informação se tornam mais difíceis. O que pode resultar em conclusões e tomadas de decisão imprecisas por parte dessas aplicações (FALBO et al., 2004). Desse modo, ainda são necessários métodos mais eficientes para gerenciar os dados gerados pelo monitoramento em uma nuvem e assim obter uma visão abrangente da mesma de forma rápida e contínua (ACETO et al., 2013).

Sistemas de monitoramento precisam ser moldáveis ao ambiente virtual, sendo suportados por uma plataforma que possa coletar, processar e disseminar informações que definam a rede. Visto que, em uma rede virtual, a topologia de recursos pode mudar dinamicamente na medida que elementos podem ser adicionados ou removidos a qualquer momento (CLAYMAN et al., 2011). Sistemas de gerência, como o *Nagios* (NAGIOS CORE), têm abordado o monitoramento de grandes sistemas distribuídos. No entanto, eles são projetados para uma infraestrutura física, onde há pouca dinamicidade (CLAYMAN et al., 2011). Nesse

contexto, o mecanismo para se extrair os dados da infraestrutura é relativamente simples, podendo ser feito de forma manual. Porém, em um ambiente em nuvem, essa prática precisa ser automatizada.

Através do levantamento feito em (FATEMA et al., 2014), algumas das principais ferramentas enfrentam problemas na descoberta automática de novos recursos, prejudicando a identificação da topologia de recursos, e, assim, acabam inviabilizando o monitoramento em ambientes virtuais.

1.1 Objetivos

O objetivo geral deste trabalho é implementar um *middleware* para a orquestração da gerência de recursos em um ambiente virtual. Para tal, é desenvolvida uma aplicação que dentro de um ambiente virtual possa fazer o levantamento da topologia de recursos, dispondo os dados coletados em um banco de dados orientado a grafo. A partir dessa informação, é feita a orquestração de ferramentas, recursos e dados de gerência, ou seja, é possível realizar a montagem do sistema de gerência. Esse objetivo geral pode ser detalhado nos seguintes objetivos específicos:

- Construir dentro do *Openstack*, plataforma de IaaS utilizada como exemplo, uma aplicação que leia os dados de registro das instâncias de recursos;
- Gerar uma topologia de recursos em um banco de dados com semântica voltada para topologias (grafos), como é o caso do *Neo4j*;
- Integrar o *Nagios* ou qualquer *software* convencional de gerência de rede ao *middleware,* melhorando-lhe a capacidade de distinguir os recursos monitorados e suas interdependências;

1.2 Base para o Desenvolvimento do Trabalho

Este trabalho, apesar de abordar uma área recente da Computação, contou com a colaboração de disciplinas estudadas durante a graduação, a saber:

- Banco de Dados - fundamental para o trabalho com a base de dados do

Openstack.

- Sistemas Operacionais - primeira matéria a abordar o assunto virtualização.
- Redes de Computadores - base para todo o projeto, foi importante para lidar com ferramentas de monitoramento.
- Desenvolvimento Web e Web Semântica - introduziu o tema *Linked Data* e a importância do uso da semântica nas aplicações.
- Teoria dos Grafos - colaborou no entendimento da estrutura de dados grafo, apesar deste trabalho não exigir conceitos complexos.

1.3 Organização do Texto

Esta monografia é estruturada em cinco capítulos e contém, além da introdução exposta, os seguintes capítulos:

- *Capítulo 2 – Computação em Nuvem, Gerência de Recursos e Banco de Dados Orientado a Grafo*: aborda brevemente temas relevantes ao contexto do trabalho.
- *Capítulo 3 – Orquestração da Gerência de Recursos*: expõe o problema e a solução do mesmo.
- *Capítulo 4 – Desenvolvimento da Orquestração da Gerência de Recursos*: trata o desenvolvimento da solução proposta.
- *Capítulo 5 – Considerações Finais*: apresenta as conclusões obtidas neste trabalho.

1.4 Trabalhos Correlatos

O *Openstack* (OPENSTACK) possui o *Ceilometer* como um módulo opcional que provê serviços de coleta de métricas da utilização de recursos físicos e virtuais. No entanto, o propósito do módulo é coletar dados de recursos básicos para fins de faturamento e cobrança. Este trabalho propõe um passo a mais, visto que o dado módulo pode ser integrado ao *middleware* como uma instância dos sistemas de gerência, e, assim, cumprir seu propósito.

Em (CALERO et al., 2015) é feita a integração do *Nagios* ao *Openstack.* O trabalho propõe importantes avanços ao tema, como o monitoramento sem o uso de agentes. Porém, ainda assim, os dados gerados estão desintegrados em múltiplas VMs que possuem o papel de gerência no sistema.

Por fim, em (KANG et al., 2014) é utilizada uma abordagem similar à utilizada neste trabalho, no que tange a integração dos dados da nuvem. Contudo, o trabalho, descoberto durante a realização deste, não possui uma solução específica para a gerência e monitoramento do ambiente em nuvem.

2 COMPUTAÇÃO EM NUVEM, GERÊNCIA DE RECURSOS E BANCO DE DADOS ORIENTADOS A GRAFO

Este capítulo aborda brevemente temas relevantes ao contexto deste trabalho, a saber: computação em nuvem, banco de dados orientados a grafo e gerência de recursos.

2.1 Computação em Nuvem

Computação em nuvem é uma área da computação em evidência e, por isso, tem sido bastante discutida. Em geral, pode-se resumir o termo computação em nuvem à utilização de *hardware* e *software* do *datacenter* de forma que tais recursos computacionais sejam providos de forma transparente, com um esforço mínimo e que haja uma interação entre os serviços oferecidos e seus usuários. Em (MELL; GRANCE, 2011), define-se computação em nuvem dentre suas características, modelos de serviço e modelos de desenvolvimento.

As características básicas que definem a computação em nuvem são:

- Serviço automático sob demanda - Os serviços devem ser providos sem interferência humana, de forma automatizada.
- Amplo acesso a rede - Os recursos devem estar disponíveis na rede e devem ser multiplataforma.
- Conjunto de recursos - Todos os recursos, ou seja: armazenamento, processamento e memória, devem ser associados dinamicamente.
- Elasticidade dos recursos - Os recursos devem ser escaláveis e assim parecer infinitos aos olhos do consumidor.
- Métrica de serviços - Os serviços providos precisam possuir um alto grau de otimização, sendo devidamente controlados e monitorados.

O uso da nuvem acabou criando uma infinidade de modelos de serviços. Todos os termos são generalizados através do termo XaaS. No entanto, tais modelos são bem representados e resumidos por três padrões. A saber:

- SaaS - Aplicações em nuvem onde apenas alguns ajustes nas configurações

são permitidos. A liberdade da customização do ambiente estará limitada pela aplicação e seu respectivo fornecedor.

- PaaS - Neste modelo um ambiente de desenvolvimento é oferecido ao usuário. Em relação ao SaaS, os contratantes desse tipo de serviço podem contar com mais recursos para controlar a plataforma, dada as limitações de cada provedor.
- IaaS - Para este serviço os recursos de *hardware* e *software* são de inteira responsabilidade do usuário. Em uma IaaS pode-se gerenciar recursos básicos como: armazenamento, processamento, memória e sistema operacional.

Os serviços da computação em nuvem podem ser oferecidos de diversas formas. São modelos de desenvolvimento da computação em nuvem:

- Nuvem privada - A nuvem privada é, geralmente, destinada a uma organização. Onde a mesma pode ser gerenciada pela própria entidade ou por um terceiro.
- Nuvem comunitária - Esse tipo de nuvem é destinado a um grupo específico com interesses em comum. As regras dentro dessa nuvem devem estar bem amarradas e a gerência fica com uma ou mais organizações envolvidas, terceiros ou uma composição entre ambos.
- Nuvem pública - É o tipo de nuvem provido para uso geral. Aqui emprega-se a prática *pay-as-you-go*, ou seja, pague pelo que você necessita. Uma nuvem pública normalmente é definida por empresas com finalidades comerciais possuindo grandes *datacenters* ou ainda por instituições governamentais e sem fins lucrativos.
- Nuvem Hibrida - Nesse tipo de nuvem, como o nome supõe, mescla-se dois ou mais dos tipos acima. As organizações envolvidas nesse tipo de nuvem possuem aplicações padronizadas e portáveis em comum.

2.1.1 Openstack

Nesse contexto, o *OpenStack* (OPENSTACK) atua como uma plataforma de

código aberto que provê uma IaaS e dá suporte a todos os outros modelos de desenvolvimento da computação em nuvem. Esse sistema é projetado para suprir qualquer tipo de demanda, atuando no controle de recursos computacionais, de rede e de armazenamento, como mostra a Figura 1. Na maioria dos casos o *Openstack* é utilizado para uso geral, no entanto, a aplicação pode ser utilizada para fins de computação de alto desempenho, alta carga de trabalho e análise de dados, redes de alta performance, aplicações distribuídas e escaláveis.

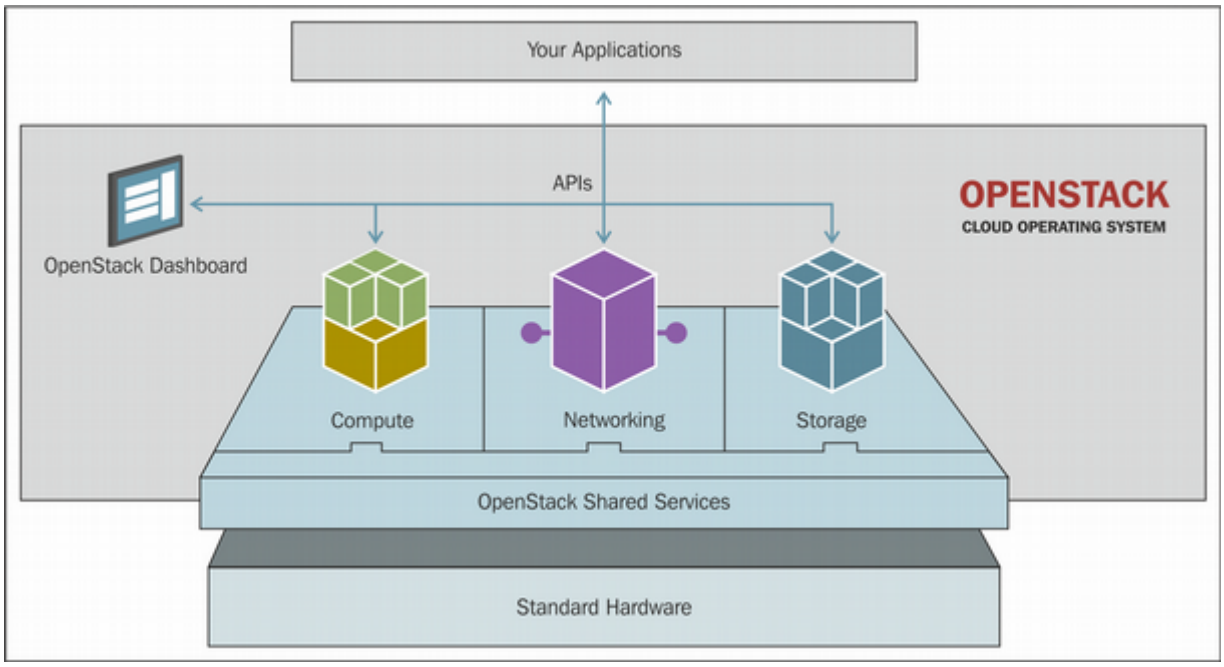

Figura 1: Visão Geral do *Openstack* (OPENSTACK).

A tecnologia por trás da plataforma consiste de vários projetos inter-relacionados. Cada componente dispõe de uma API aberta que dá a possibilidade de gerenciar recursos através de comandos, outras aplicações suportadas e interface web. Alguns desses serviços são obrigatórios:

- *Keystone* - Provê a autenticação e autorização para os demais serviços.
- *Glance* - Gerencia as imagens, ou seja, modelos de instância cujo estado do sistema foi capturado em determinado tempo. Esses modelos também são chamados de *Snapshots*.
- *Nova* - É a maior parte do sistema, pois administra o ciclo de vida das instâncias virtuais do ambiente, sendo responsável por todas as tarefas computacionais como criação e remoção dessas. Além disso, faz o controle dos *hypervisors*, que administram os recursos de *hardware* da máquina física a fim de otimizar o desempenho dos nós virtuais.
- *Neutron* - Fornece aos outros serviços a conexão à rede. Suporta vários componentes e tecnologias, inclusive as mais recentes, como o SDN.
- *Swift* - Módulo que dispõe o armazenamento de objetos, possibilitando o *Backup* do ambiente. Objetos, ao contrário de arquivos, não são estruturados e correspondem a unidades de armazenamento.

- *Cinder* - Concede locais de disco, ou seja, sistemas de arquivo para as máquinas virtuais.

Além desses componentes, há outros serviços em estado maduro de desenvolvimento:

- *Horizon* - Coordena os componentes do *Openstack* através de uma interface Web.
- *Heat* - Responsável pela orquestração da infraestrutura da nuvem, fornecendo modelos de topologias de recursos previamente configuradas.
- *Manila* - Baseado no projeto do serviço *Cinder*, provê sistemas de arquivos compartilhados.

Na Figura 2, percebe-se de forma mais clara o papel de cada serviço do *Openstack*.

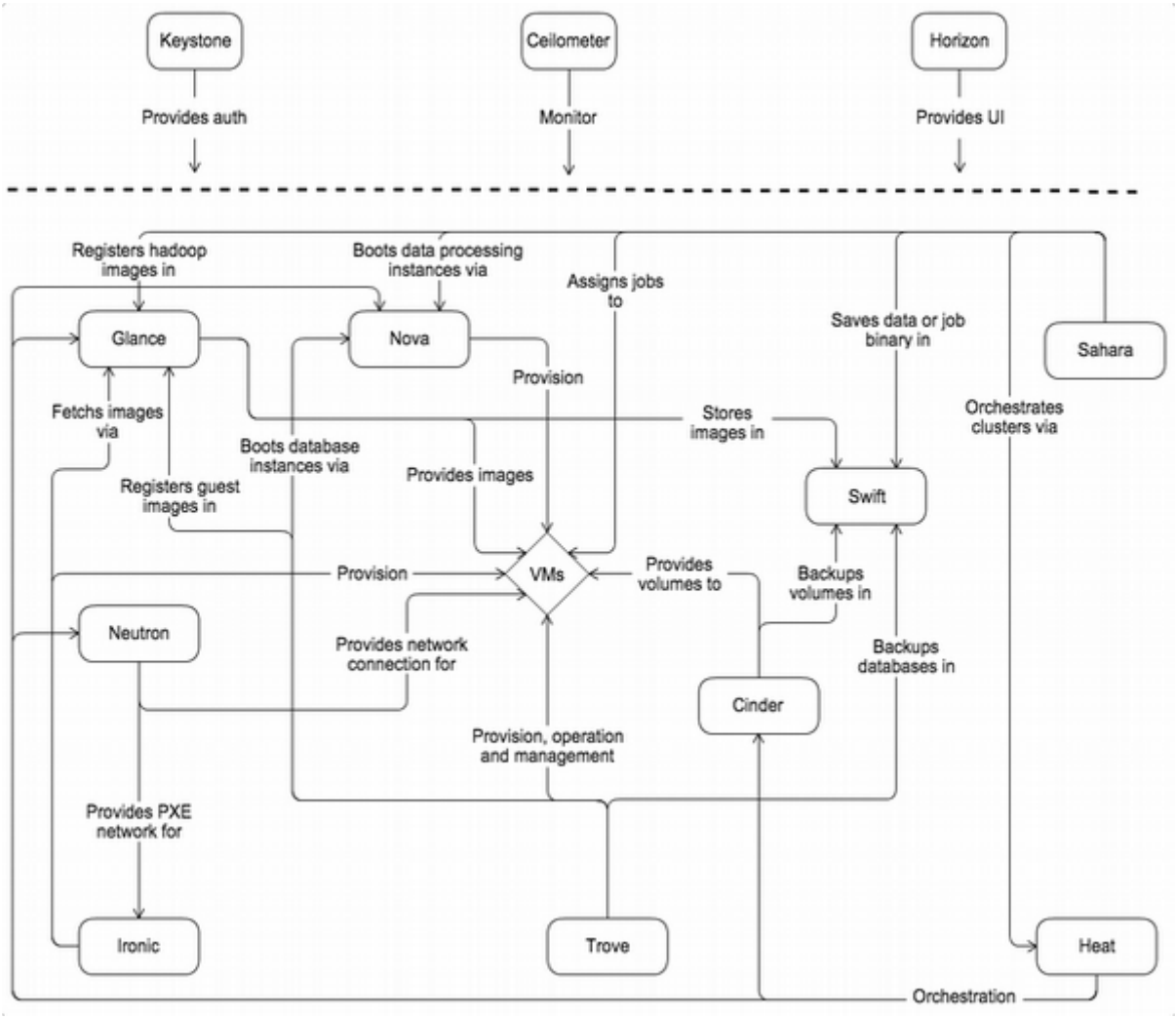

Figura 2: Arquitetura Conceitual do *Openstack* (OPENSTACK).

Em um ambiente virtual o hardware sempre deve ser planejado para suportar a carga planejada. Do mesmo modo, o sistema operacional e o *hypervisor* precisam ser analisados em relação as questões de suporte e compatibilidade. Por fim, como os serviços do *Openstack* requerem acesso ao banco de dados para gravar o estado da aplicação e informações de configurações, deve-se fazer uma escolha por um SGBD capaz de executar os *drivers* do *SQLAlchemy*, biblioteca de mapeamento objeto-relacional escrita na linguagem de programação *Python*.

O *Openstack* é altamente configurável, por isso, existem diversas possibilidades de instanciação de sua arquitetura. Basicamente, o que varia de uma instância para outra é a disposição dos componentes em várias máquinas físicas, as quais são chamadas de nós. Dentro dessa divisão, encontram-se:

- Controlador - Responsável por executar os serviços que irão administrar o ambiente. Desse modo, contêm o painel de controle, os serviços de identidade, de imagem e, porções de serviços de rede e computação. Ou seja, os serviços *horizon*, *keystone*, *glance* e partes do *neutron* e *nova*, respectivamente.
- Computação - Executa as instâncias do ambiente e o *hypervisor*. Desse modo, nesse nó também se encontram módulos para garantir a conexão e segurança entre as máquinas virtuais.
- Armazenamento - Mantêm todos os dados requisitados pelo ambiente. A presença desse nó é facultativa, podendo ser anexado ao nó controlador.
- Rede - Responsável por criar toda a rede virtual do sistema, o que inclui, por exemplo, a conexão com a internet. Esse nó pode executar *switches* virtuais como o *Open vSwitch* (OPENVSWITCH).
- Utilidade - Esses tipos de nós são opcionais, e normalmente executam os serviços como: *Ceilometer* para coleta de métricas, *Trove* para o Banco de Dados, *Designate* para o DNS, dentre outros.

A Figura 3 mostra um exemplo de arquitetura e suas respectivas conexões.

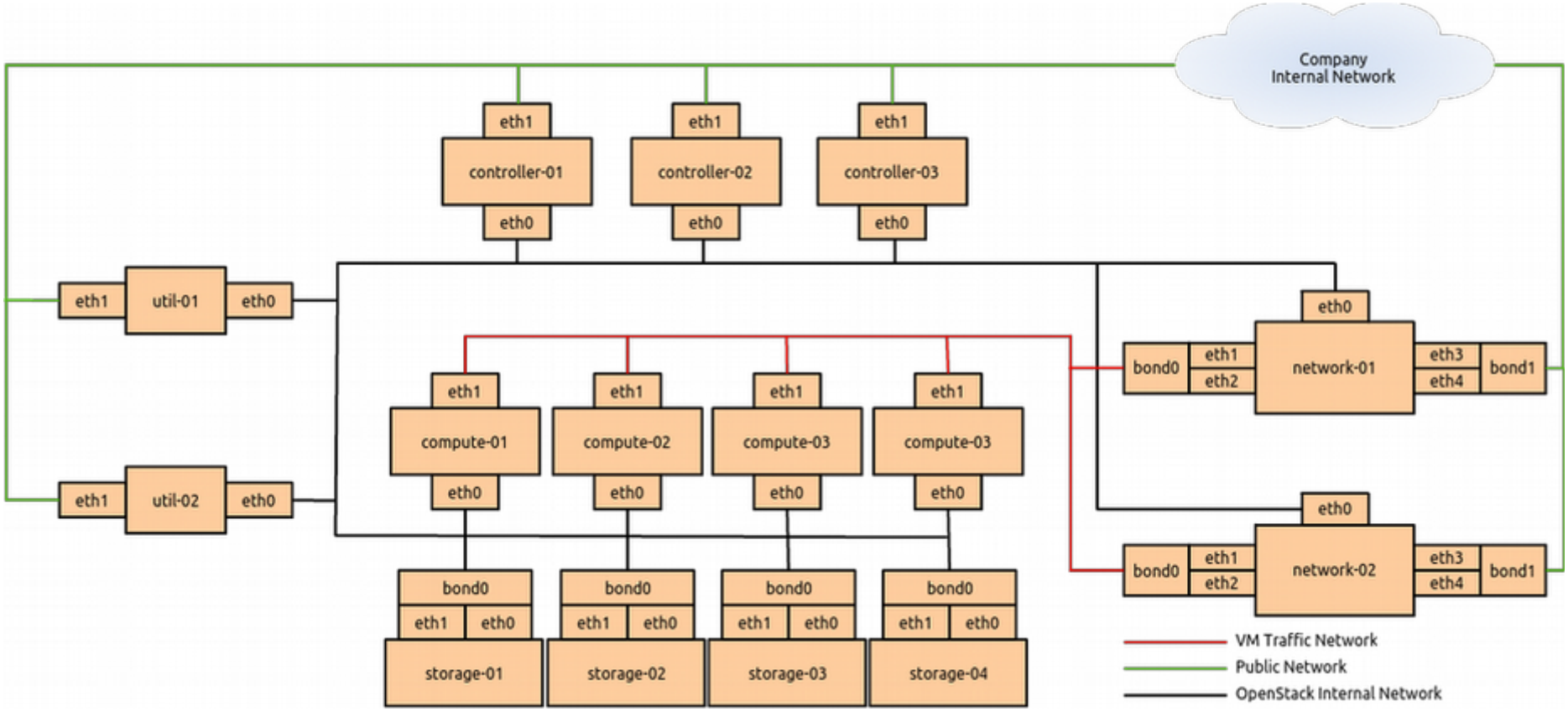

Figura 3: Exemplo de Arquitetura do *Openstack* (OPENSTACK).

2.2 Banco de Dados Orientados a Grafos

Cada proposta de modelo de banco de dados é baseada em um princípio teórico que serve como base para o seu desenvolvimento. Baseado na teoria dos grafos, um modelo de banco de dados orientado a grafos é um paradigma no qual as estruturas de dados para os objetos do banco de dados são modelados como grafos dirigidos, possivelmente rotulados. A manipulação de dados ainda deve ser expressa por operações, como: caminhos, vizinhanças, subgrafos, conectividades, dentre outros. Esse modelo deve ainda cumprir a consistência de dados através de restrições de integridade sob a estrutura do grafo (ANGLES; GUTIERREZ, 2008).

Uma infraestrutura de rede pode ser representada através de grafos, possibilitando: fazer um catálogo de ativos, visualizar a maneira que os mesmos estão sendo desenvolvidos e identificar todas as relações possíveis da rede. Desse modo, um banco de dados orientado a grafo pode complementar o trabalho da gerência e monitoramento (ROBINSON; WEBBER; EIFREM, 2015). A Figura 4 mostra um exemplo de representação de *datacenter* através de bancos de dados orientados a grafos. Com isso, a partir do grafo, é possível representar uma topologia de recursos e elementos disponíveis em uma infraestrutura.

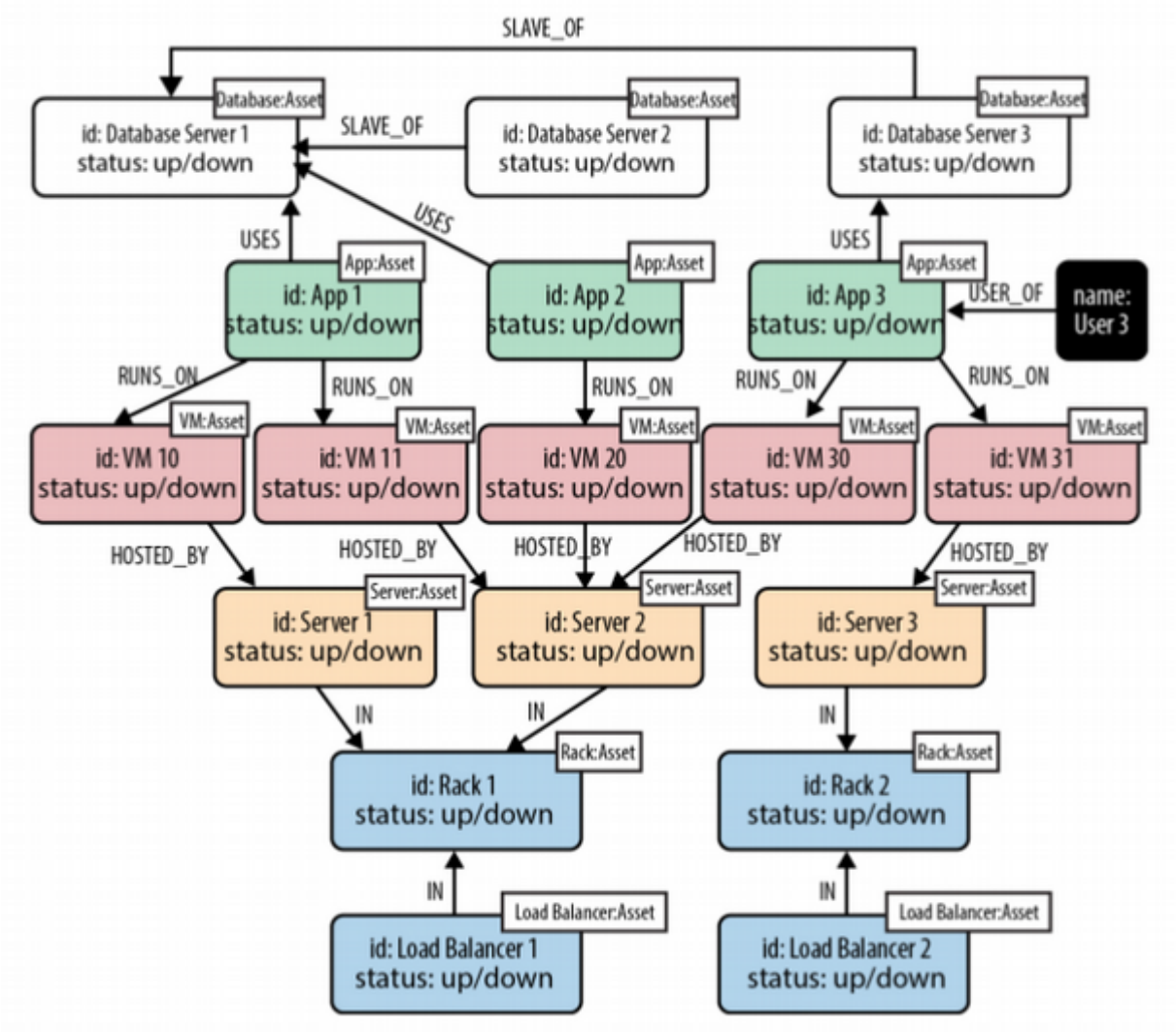

Figura 4: Exemplo de um *Datacenter* Representado Através de um Grafo (ROBINSON; WEBBER; EIFREM, 2015).

Como se pode observar na Figura 4, o grafo possui relações diretas com tipos associados, e cada nó possui um tipo e um rótulo identificando o mesmo. Esse tipo de grafo pode também ser chamado de grafo semântico (BARTHÉLEMY; CHOW; ELIASSI-RAD, 2005). Com isso, o banco de dados orientado a grafo propõe uma série de facilidades em relação aos modelos relacionais mais utilizados no mercado. As Figuras 5 e 6 exemplificam a comodidade do modelo orientado a grafo em relação aos tradicionais.

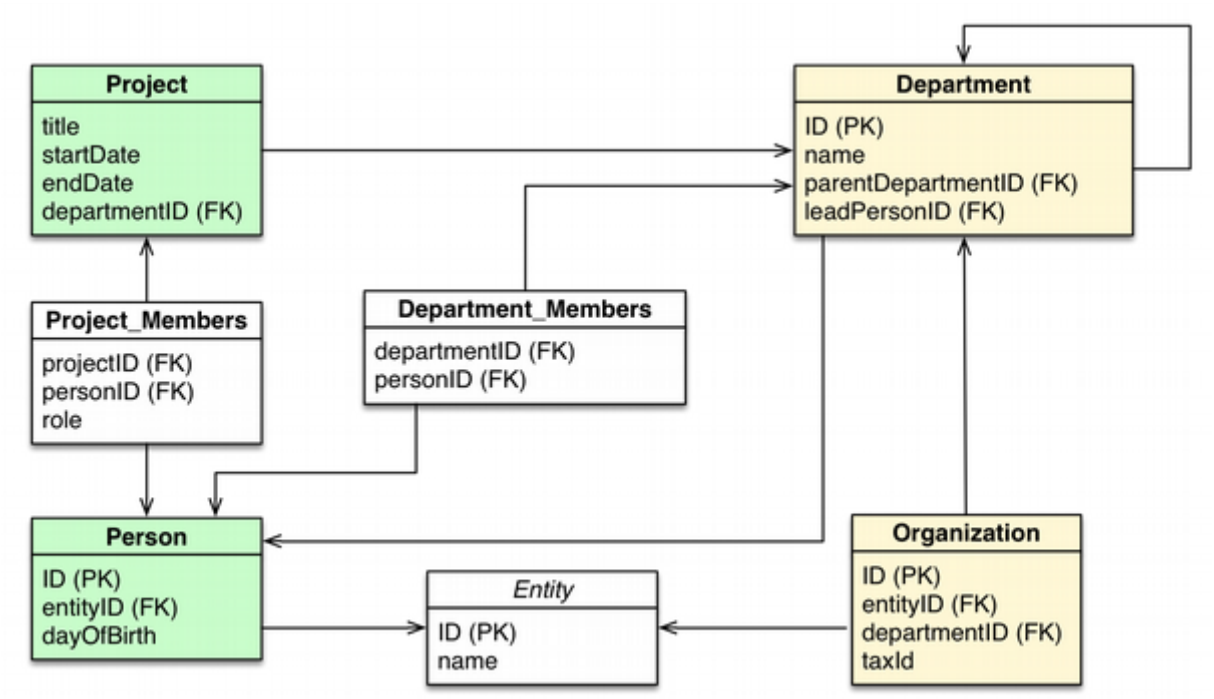

Figura 5: Representação no Modelo Relacional (NEO4J).

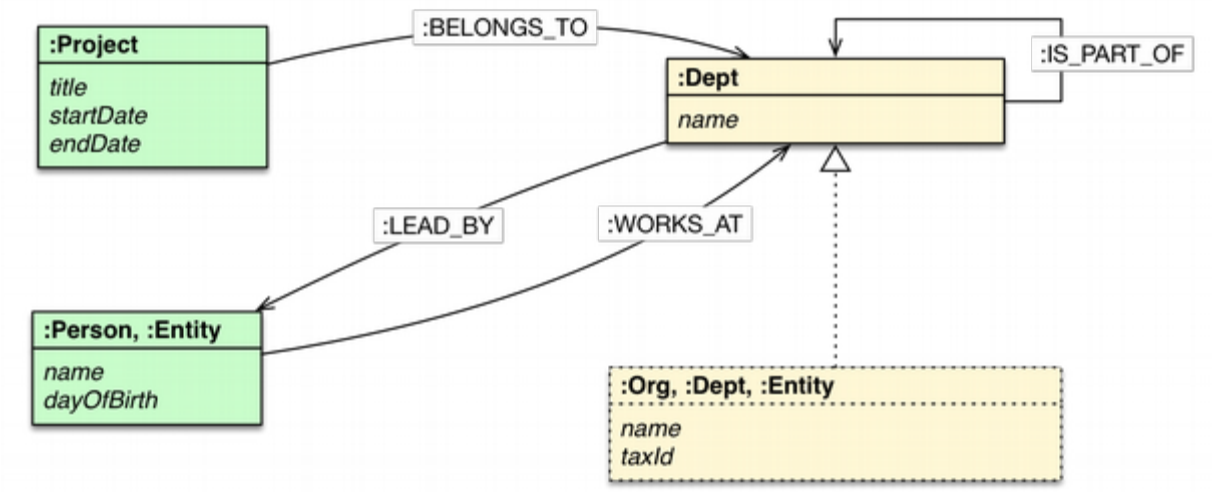

Figura 6: Representação no Modelo Orientado a Grafo (NEO4J).

2.2.1 Neo4j

O *Neo4j* (NEO4J GRAPH DATABASE) é um SGBD orientado a grafos robusto, escalável e de alta performance. O mesmo possui uma interface web para facilitar a

visualização do grafo disposto no banco de dados. Através dessa interface é possível realizar consultas, como pode ser visto pela Figura 7, atualizações e editar grande parte das configurações.

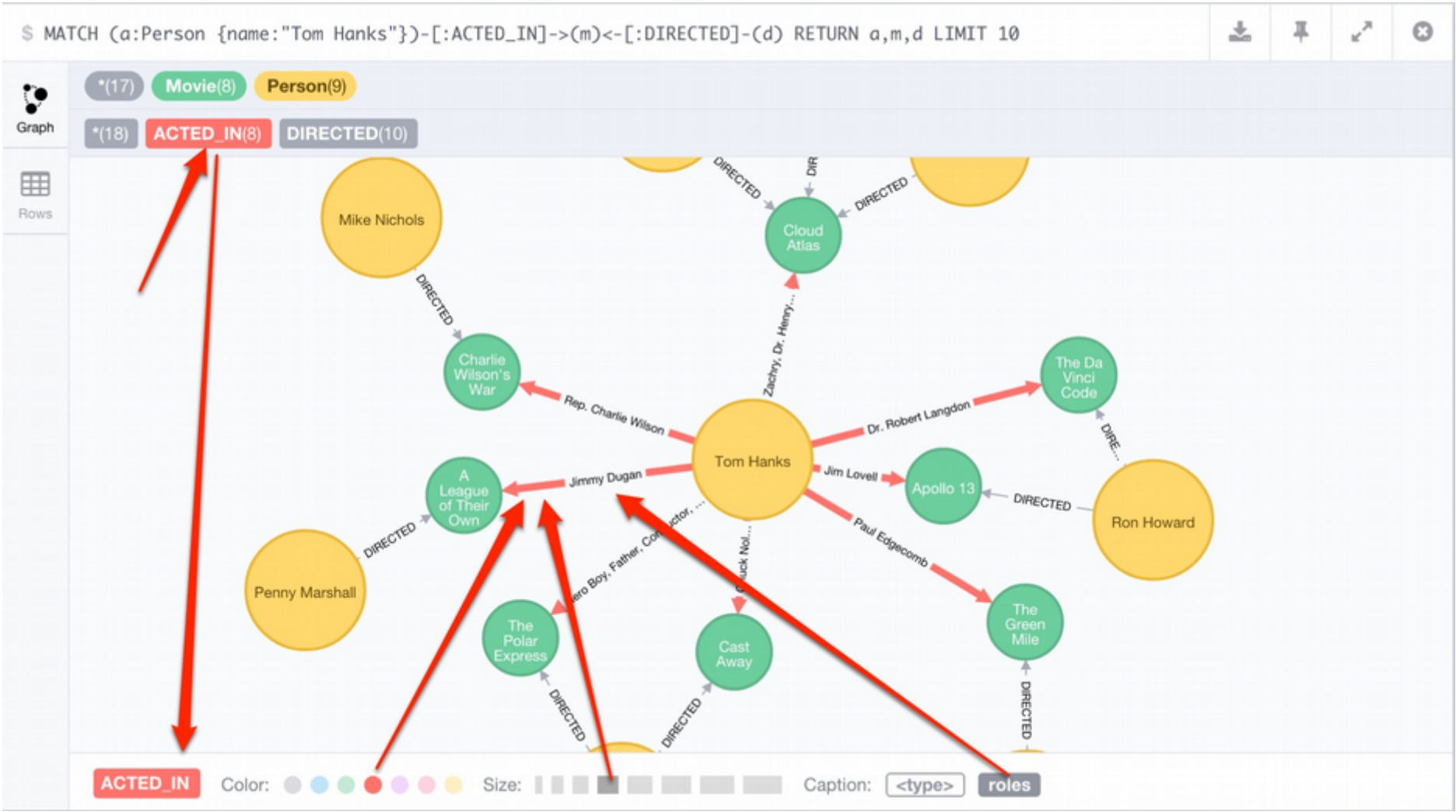

Figura 7: Dados no *Neo4j* Observados Através da Interface Gráfica (NEO4J GRAPH DATABASE).

Assim como o SQL está para os banco de dados relacionais, o *Cypher* está para o *Neo4j*. Ou seja, o *Cypher* é a linguagem de programação para gerenciar os dados no *Neo4j*. A linguagem possui inspiração em outras linguagens, expressões como *where* e *order by* derivam do SQL, por exemplo. As principais características da linguagem na representação da estrutura do grafo, são:

- As propriedades dos nós ou relacionamentos são identificadas através de um par de chaves. Por exemplo: *{título: "Matrix", ano: 1997}*, onde *título* e *ano* são as propriedades, e "*Matrix"* e *1997* são os seus respectivos valores.
- Os nós são representados através de um par de parênteses. Para acrescentar informação ao nó, adiciona-se um identificador e um rótulo. Por, exemplo: *(matrix:Filme)*, onde *matrix* é o identificador e *filme* o rótulo.
- Relacionamentos são reconhecidos por um par de traços. Da mesma forma, uma relação também pode possuir um identificador e um rótulo. Além disso, pode-se indicar a orientação do relacionamento através dos sinais maior ou menor. Por exemplo: *-[função:ATUA_EM]->* , onde *função* é o identificador e *ATUA_EM* o rótulo.

Portanto, um grafo com dois nós pode ser referenciado da seguinte forma: *(keanu:Pessoa: Ator {nome: "Keanu Reeves"}) -[função:ATUA_EM {papeis: ["Neo"] }]-> (matrix:Filme {título: "Matrix"})*. Assim, para realizar consultas e atualizações nos elementos do grafo é necessário conhecer as principais expressões da linguagem, são elas:

- *Return* - Define quais padrões da consulta que devem ser retornados.
- *With* - Permite que partes de uma consulta sejam ligados, formando um canal que recebe o insumo da primeira cláusula e depois entrega o resultado para a segunda.
- *Order By* - Especifica como a saída de outra cláusula deve ser ordenada.
- *Limit* - Estabelece o número de linhas do resultado de uma consulta.
- *Skip* - Define a partir de qual linha da saída de uma cláusula o resultado deve ser mostrado.
- *Union* - Combina o resultado de outras consultas.
- *Match* - Utilizada para reunir padrões descritos através dela.
- *Where* - Adiciona restrições aos resultados de outras cláusulas, filtrando o

resultado.

- *Count* - Conta o número de linhas do resultado de uma consulta.
- *Create* - Cria elementos, nós e relacionamentos, do grafo.
- *Merge* - Une informações de elementos, pode criar caso algum dado não exista no banco.
- *Set* - Atualiza ou cria rótulos ou propriedades dos elementos do grafo.
- *Delete* - Deleta os elementos do grafo.
- *Remove* - Remove propriedades ou rótulos dos elementos do grafo.

Com tais expressões é possível criar funções, e esquemas que podem ser índices ou restrições sobre os rótulos dos elementos do grafo.

Diante dessas informações, o *Neo4j* oferece compatibilidade com uma série de linguagens de programação, a saber: *Java*, *JavaScript*, *.NET*, *Python*, *Ruby*, *PHP*, dentre outras. Acerca da linguagem *Java*, no programa é possível contar com o apoio do módulo JDBC. Isso porque, o *Cypher* é uma linguagem de consulta parametrizável textual e que pode retornar resultados tabulares. Além disso, o banco de dados provê mais de uma forma de acesso aos dados, os quais pode-se citar: modo servidor, modo embutido e através do modo REST. Para o primeiro, utiliza-se uma URL, onde o acesso pode se concretizar através dos protocolos HTTP e HTTPS. O segundo, por sua vez, usa o caminho no sistema de arquivos do Sistema Operacional.

2.3 Gerência de Recursos

Há diversos motivos para se fazer a gestão de recursos em um ambiente computacional. Contudo, um dos principais motivos é garantir que cada parte da rede se comporte conforme o planejado e acordado. Para isso, deve-se coletar métricas da operação, tanto de infraestrutura como das aplicações (ELMROTH; LARSSON, 2009). Uma vez que em um ambiente virtual há um alto número de recursos, a gestão passa a ser um grande desafio. Portanto, o objetivo atual dos *datacenters* é encontrar uma ferramenta de gestão que seja escalável. Ao mesmo tempo, a aplicação deve manter uma precisão satisfatória e também não

sobrecarregar a rede (BARI et al., 2013).

2.3.1 Nagios

Há uma infinidade de aplicações de monitoramento, o *Nagios* (NAGIOS CORE), por exemplo, é uma das principais ferramentas de código aberto e que conta com uma comunidade ativa no mundo inteiro. O programa é capaz de monitorar desde recursos básicos como processamento e armazenamento até serviços de máquinas dispostas na rede. Além disso, o *Nagios* pode notificar os administradores através de vários métodos de comunicação. E, para uma melhor experiência no uso do programa, o usuário pode definir por observar o estado dos recursos monitorados através de uma interface web.

Diferentemente de outras aplicações, o *Nagios* não inclui mecanismos de monitoramento acoplados ao código do programa. Todo o trabalho é feito a partir dos *plugins*, que são *scripts* executáveis por linha de comando. O retorno desses *plugins* é utilizado para determinar o estado do recurso em observação e também para executar ações estipuladas pelo usuário.

A Figura 8 mostra, de forma geral, a arquitetura do *Nagios*. Como foi dito, o cerne do funcionamento do *Nagios* são os seus *plugins*, que se encontram na camada de abstração do monitoramento. Dando suporte a essa camada, figura a lógica de monitoramento, ou seja, o programa *Nagios* em si. O processo da aplicação não faz ideia do que está sendo monitorado, basicamente sua função é alterar o estado do item em questão. Como alvo de toda a aplicação aparecem os serviços e máquinas em análise.

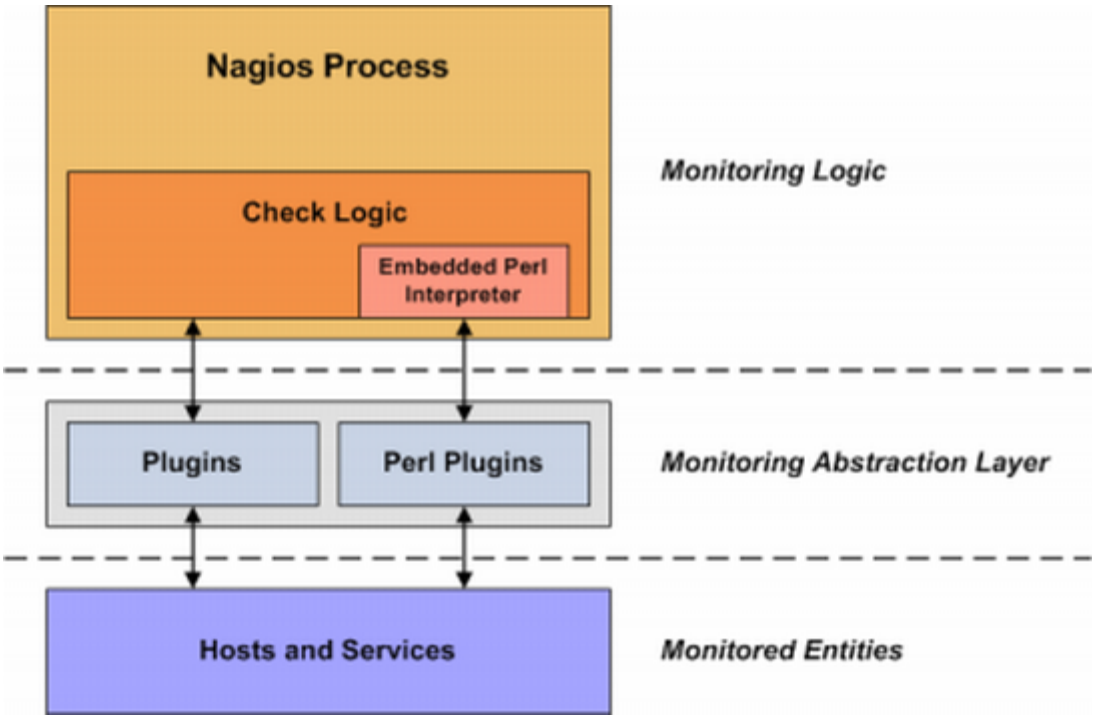

Figura 8: Arquitetura do *Nagios* (NAGIOS CORE).

De acordo com a Tabela 1, os *plugins* do *Nagios* podem retornar quatro tipos de estado dos objetos monitorados: *ok*, *warning*, *unknown* e *critical*. Dado que um recurso em pleno funcionamento (*UP*) retorna *ok*, as demais opções apontam alguma falha na comunicação. No entanto, caso o retorno seja *warning*, é sinal que o item monitorado esteja enfrentando problemas, mas na maioria das vezes está ativo. Em caso de falha de máquinas, o *Nagios* ainda investiga se somente a mesma está inativa (*DOWN*) ou se está inalcançável (*UNREACHABLE*) através de uma hierarquia definida nos arquivos de configuração. Logo, uma máquina está inalcançável se nenhum nó pai possui conexão, ou seja, esteja também inativo ou inalcançável.

Tabela 1: Possíveis Retornos dos *Plugins* (NAGIOS CORE)

Retorno do *Plugin*	Estado Preliminar
OK	*UP*
WARNING	*UP* ou *DOWN*
UNKNOWN	*DOWN*
CRITICAL	*DOWN*

2.3.2 Nmap

Além de descobrir máquinas em uma infraestrutura, também é importante encontrar os serviços que cada uma hospeda. O *Nmap* (NMAP SECURITY SCANNER) é uma ferramenta de código aberto para exploração de rede e auditoria de segurança. A aplicação utiliza pacotes TCP/IP da família de Protocolos da Internet para determinar: o estado da máquina, o tipo do Sistema Operacional, os serviços hospedados,o modelo das aplicações, dentre outras características.

O *Nmap* é utilizado via linha de comando e através de suas opções há uma infinidade de retornos possíveis. Assim, o usuário pode definir investigar um número aleatório de dispositivos ou instâncias específicas dentro da rede. Isso faz com que o *NMap* seja utilizado para diversos fins, que vão desde a administração básica de uma rede interna até a análise de vulnerabilidades no *firewall*. Geralmente, a descoberta de máquinas é atribuída ao comando *ping*, mas ela vai muito além dos simples pacotes ICMP. Os usuários podem enfrentar a rede com combinações arbitrárias de sondagens multi-portas através dos protocolos TCP, UDP e ICMP.

A principal proposta do *Nmap* é constatar os serviços de uma máquina através do escaneamento de portas. Mais uma vez, há inúmeras formas de fazer essa investigação, sendo que a maioria só está disponível para usuários com privilégios de administrador. No entanto, é possível cumprir princípios básicos da gestão de redes a partir de usuários sem tal permissão. Uma vez que uma porta é encontrada, o programa faz consultas mais precisas para averiguar qual serviço está sendo executado. A partir desses dados faz-se uso de um banco de dados com informações de aplicações e Sistemas Operacionais a fim de dar uma resposta precisa.

O resultado de todo esse processamento descrito pode ser retornado de diferentes maneiras. No entanto, a resposta comum é a saída interativa enviada à saída padrão. Porém, o retorno por XML, é uma das saídas mais importantes. Pois, através desse tipo de arquivo, outros programas podem analisar os dados de forma mais fácil e ágil.

3 ORQUESTRAÇÃO DA GERÊNCIA DE RECURSOS

Este capítulo está organizado da seguinte forma: a Seção 3.1 apresenta a definição do problema abordado neste trabalho; e, por fim, a Seção 3.2 descreve a contribuição deste trabalho na solução do problema exposto.

3.1 Definição do Problema

Como foi mencionando no capítulo introdutório deste trabalho, o monitoramento em uma nuvem enfrenta os seguintes desafios:

- Cada aplicação necessita entender a topologia de recursos dinâmica da nuvem. Visto que, em uma rede virtual, os recursos podem ser adicionados ou removidos a qualquer momento (CLAYMAN et al., 2011). Assim, as ferramentas necessitam prover uma precisão no reconhecimento dos recursos da infraestrutura.
- As ferramentas precisam realizar inferências sobre os recursos da nuvem. Entretanto, ainda são necessárias melhores táticas para gerir a quantidade de dados gerados através do monitoramento e com isso obter uma visão ampla da nuvem de forma eficaz (ACETO et al., 2013). Logo, o resultado do monitoramento de todas as ferramentas precisa ser discernido de uma forma integrada.
- Os sistemas devem realizar a descoberta da nuvem de forma automatizada. Por meio do estudo realizado por (FATEMA et al., 2014), algumas das principais ferramentas enfrentam problemas na detecção automática de novos recursos, prejudicando a identificação da topologia de elementos.

3.2 Solução do Problema

Uma infraestrutura de rede pode ter seus elementos representados através de grafos, possibilitando: fazer um catálogo de ativos, visualizar a maneira que os

mesmos estão sendo desenvolvidos e identificar todas as relações possíveis da rede. Para a gerência e análise de uma rede, uma solução de banco de dados orientado a grafos permite também o suporte tanto para provedores como para consumidores da nuvem. Pois dadas as relações, é possível selecionar quais recursos são cabíveis ao administrador da nuvem e quais são de responsabilidade do usuário (ROBINSON; WEBBER; EIFREM, 2015). No ponto de vista do administrador e do usuário da nuvem é possível entender as dependências da rede, ou seja, identificar máquinas físicas e virtuais, facilitando a gestão de questões comuns ao ambiente virtual como a migração de recursos. No ponto de vista das ferramentas de monitoramento, onde cada uma precisa gerar sua própria topologia de recursos, uma abordagem semântica de um banco de dados orientado a grafos dispensa o trabalho de descobrir elementos em uma rede e também facilita a identificação de relações entre os recursos.

A Figura 9 mostra a interação entre o *middleware* desenvolvido a fim de solucionar o problema, as plataformas IaaS e as ferramentas de monitoramento.

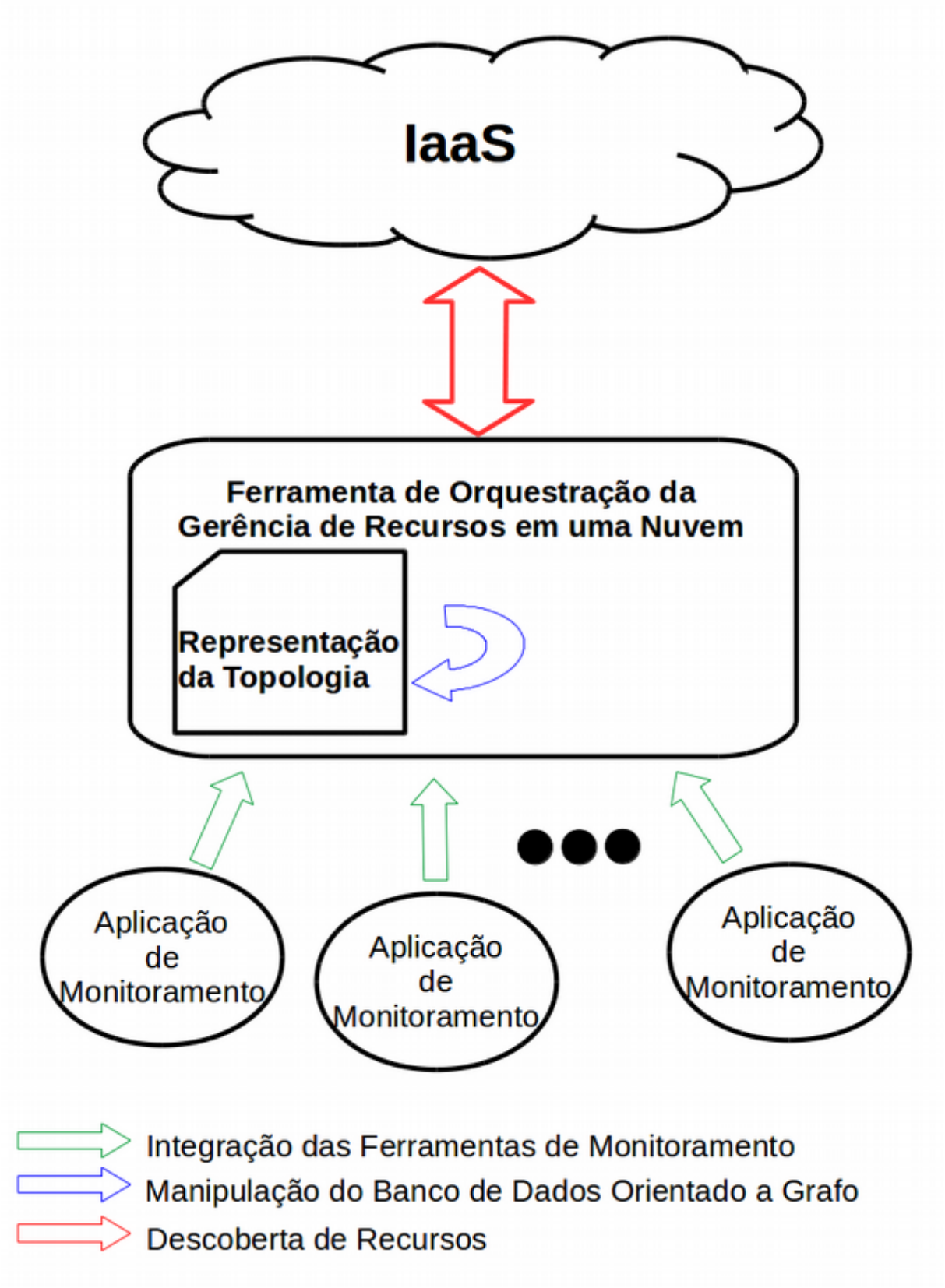

Figura 9: Arquitetura da Solução Proposta.

Nesta seção é proposta uma abordagem de orquestração da gerência de uma nuvem, ou seja, uma solução para a montagem e modelagem do sistema de gerência. A peça chave para a orquestração é o grafo semântico, que entrega uma referência da topologia de recursos para que ferramentas de gerência possam desempenhar o seu trabalho, cumprindo as questões levantadas em (ACETO et al., 2013), citadas na seção anterior. O processo de orquestração segue os seguintes passos: descoberta de recursos, manipulação do banco de dados orientado a grafo e integração das ferramentas de monitoramento. A Figura 10 mostra, através do diagrama de pacotes, a relação entre os passos.

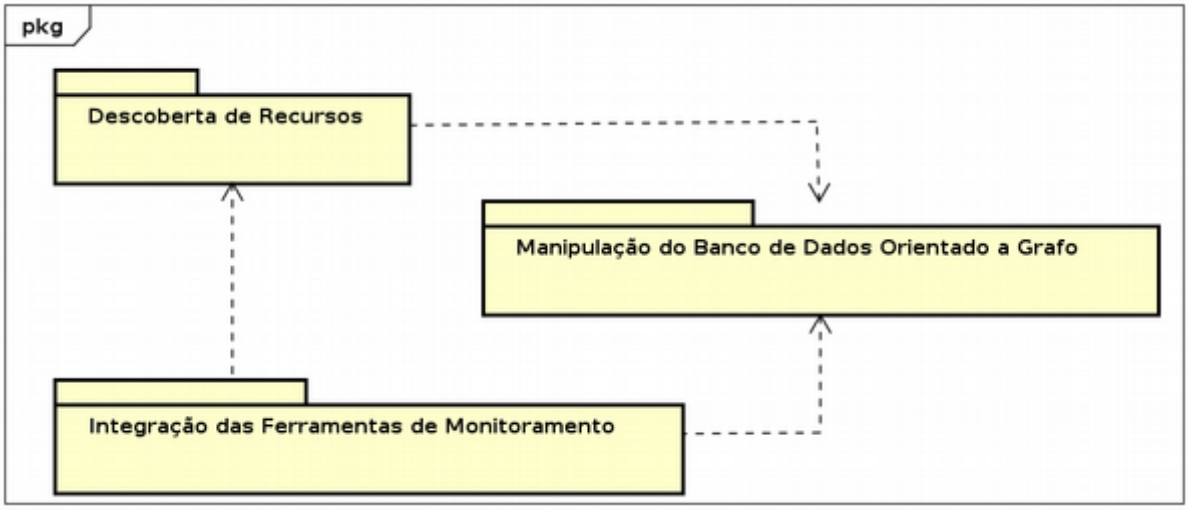

Figura 10: Diagrama de Pacotes.

3.2.1 Descoberta de Recursos

A descoberta de recursos é um ponto de suma importância na proposta. O método para encontrá-los pode variar dentre as diversas soluções, desde as mais simples até as mais eficientes. Pode-se utilizar, por exemplo, ferramentas que utilizam o protocolo ICMP, no entanto, neste trabalho a descoberta da topologia de recursos é realizada através da leitura de banco de dados providos pelas IaaS. A base de dados provê informações básicas acerca do ambiente físico onde as máquinas virtuais executam, bem como dados mais completos de cada instância. Feito o trabalho no banco de dados, são capturadas as aplicações respectivas a

cada máquina virtual.

Dentre as soluções de código aberto para uma plataforma IaaS, pode-se citar: *Apache CloudStack* (APACHE CLOUDSTACK), *OpenNebula* (OPENNEBULA) e *Eucalyptus* (EUCALYPTUS). No entanto, para este trabalho foi escolhido o *Openstack*, ferramenta consolidada que conta com uma comunidade de desenvolvedores relevante.

3.2.2 Manipulação do Banco de Dados Orientado a Grafo

Esse módulo do sistema deve prover funções de consulta de escrita e leitura para a manipulação do grafo. O banco de dados orientado a grafo é preenchido com as informações da nuvem coletadas pelo módulo de *Descoberta de Recursos*. Assim, cria-se a topologia de recursos e estabelece-se todas as dependências, ou seja, determina-se o nó físico, nó virtual e seus respectivos serviços. As principais informações capturadas acerca das máquinas seguem a proposta de (MONTEIRO et al., 2014) e podem ser observadas através da Figura 11.

Devido a dinamicidade do ambiente virtual, é importante reconhecer os recursos através de um UUID. Além de facilitar a distinção dos recursos, isso se torna necessário para o apoio de operações comuns em ambientes virtuais, como a migração.

As seguintes informações do nó de computação (*Compute*) são gravadas no grafo: nome (*Name*), protocolo IP (IP), a representação do contexto do nó (*Layer*), o UUID (uuid). Para uma instância (*Instance*), o grafo mantém, além das informações do controlador, as seguintes características: o MAC associado à Instância (MAC),a interface de rede associada ao IP (Interface), o UUID do projeto (*Project*), o UUID do locatário (*Tenant*) e o UUID da interface associada ao roteador associado a instância (*Router*). Já um serviço (*Service*) contêm nome (Name) e UUID (uuid), obrigatoriamente. E, opcionalmente, podem conter uma a lista de propriedades monitoradas(*Properties*) e uma lista de valores associados as propriedades coletadas (*Values*). Cada serviço pode possuir um atributo em um nó a parte do grafo para definir seu estado (*ServiceStats*). Esse nó apenas contém o UUID do serviço como atributo e uma variável indicando se o serviço monitorado sofreu

alguma alteração. Por fim, para conectar os nós são utilizadas as seguintes relações: *Hosts* e *Provides*, ou seja, um nó controlador hospeda (*Hosts*) várias instâncias que oferecem (*Provides*) também vários serviços.

É importante salientar que tanto o nó ServiceStats, como as características dos Serviços não possuem relevância para este trabalho. Estas funcionalidades são utilizadas no trabalho de graduação do aluno de Ciência da Computação da Universidade Federal do Espírito Santo, Káio César Ferreira Simonassi[1].

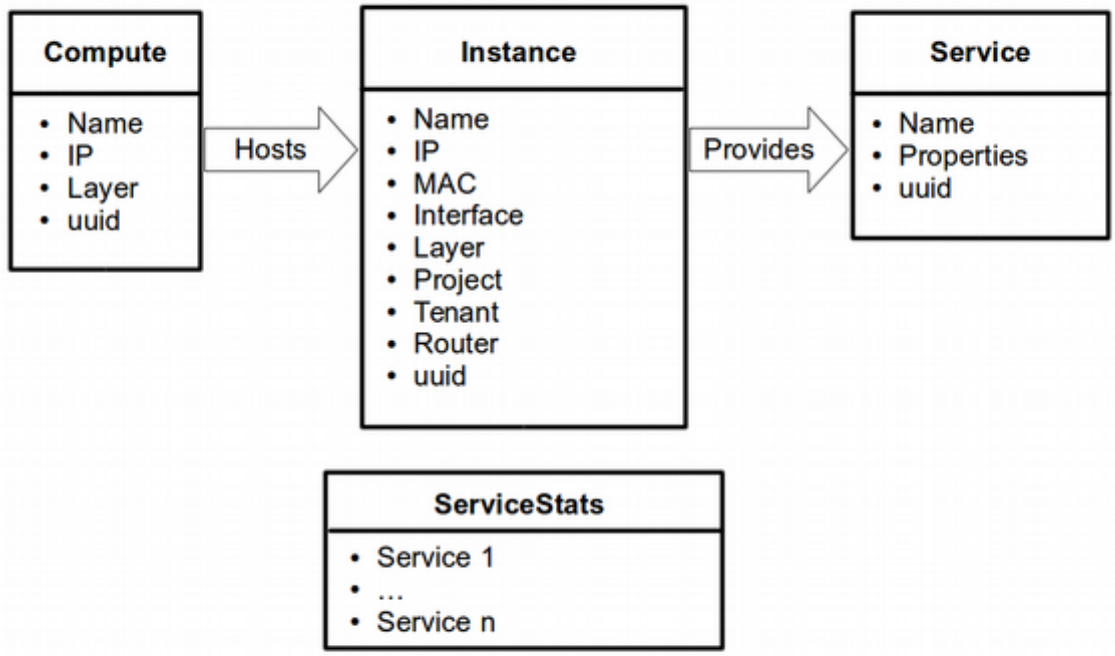

Figura 11: Modelo de Dados do Grafo Semântico.

Sistemas de banco de dados orientado a grafo são aplicações recentes. No entanto, este trabalho utilizou o *Neo4j*, ferramenta de código aberto mais popular no segmento.

3.2.3 Integração das Ferramentas de Monitoramento

1 Disponível em: <http://github.com/ksimonassi/projetoGraduacao>. Acesso em 30/06/2016.

A partir do momento que a topologia de recursos está disponível, instâncias dos sistemas de gerência podem desempenhar sua função sem o dever de realizar a descoberta de elementos na rede. Com isso, só há a necessidade implementar consultas ao banco de dados orientado a grafo e então monitorar os elementos pretendidos. Nesse momento, o administrador e usuários do ambiente virtual podem de fato realizar a orquestração da gerência. Ou seja, a partir de agora pode se definir quais ferramentas irão monitorar os recursos, quais elementos serão monitorados e que tipo de dado será inspecionado. Essas funcionalidades são estabelecidas como casos de uso do sistema, como se pode observar através da Figura 12.

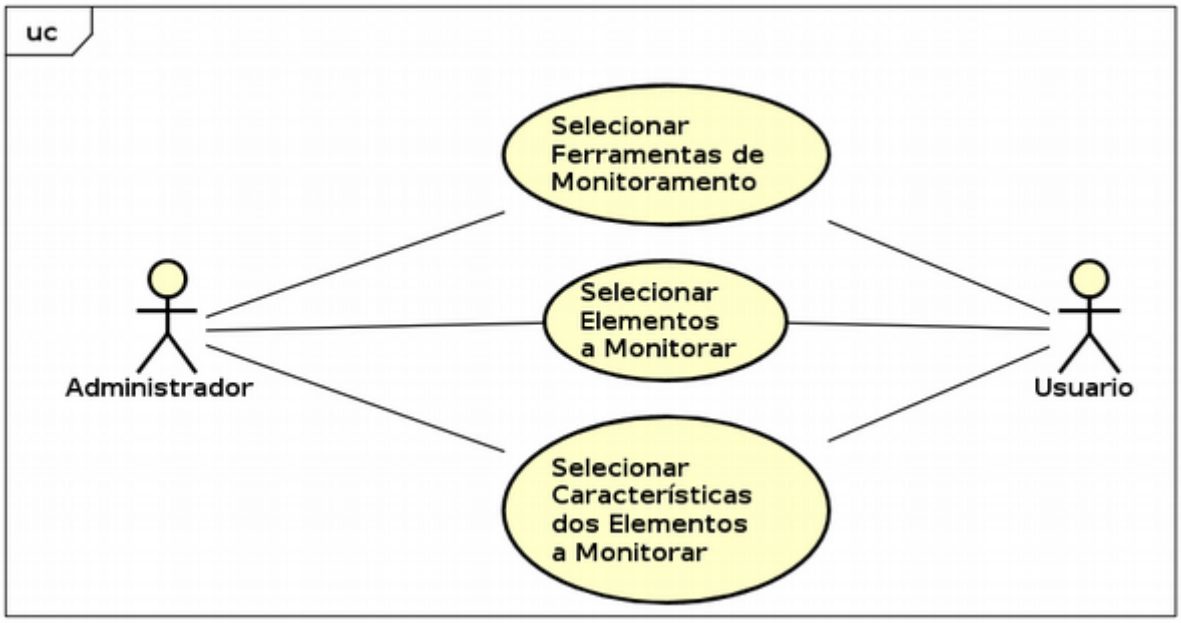

Figura 12: Casos de Uso do Sistema.

Acerca dos casos de uso do Sistema, pode-se detalhar:

- Selecionar Ferramentas de Monitoramento - Acerca das diversas ferramentas de monitoramento integradas ao sistema, os usuários podem alocar a que melhor se comporta ao recurso que pretendem monitorar.
- Selecionar Elementos a Monitorar - Uma vez que foi selecionado qual ferramenta de monitoramento utilizar, pode-se selecionar o tipo de recurso a observar. Assim, pode-se escolher desde elementos físicos a elementos virtuais.

- Selecionar Características dos Elementos a Monitorar - Cada elemento da topologia de recursos possui uma diversidade de características. No entanto, os usuários do sistema podem selecionar quais devem ser gerenciados.

O *Nagios* é uma das ferramentas de código aberto para o monitoramento de sistemas mais utilizadas no mundo. Este trabalho utilizou a aplicação como exemplo de integração.

4 DESENVOLVIMENTO DA ORQUESTRAÇÃO DA GERÊNCIA DE RECURSOS

Este capítulo está organizado da seguinte forma: a Seção 4.1 apresenta o desenvolvimento da solução do problema em termos de implementação; a Seção 4.2 descreve o ambiente de teste; e, por fim, a Seção 4.3 aborda as dificuldades encontradas na elaboração da proposta deste trabalho.

4.1 Desenvolvimento da Solução do Problema

De acordo com o capítulo anterior, os componentes da arquitetura deste trabalho, são: descoberta de recursos, manipulação do banco de dados orientado a grafo e integração das ferramentas de monitoramento. Os tópicos a seguir irão descrever cada componente de acordo com a sua implementação, descrevendo a funcionalidade de cada classe, como pode ser visto através do diagrama da Figura 13.

A aplicação foi desenvolvida na linguagem *Java* e se encontra disponível no repositório do *GitHub*[2].

2 Disponível em: <https://github.com/orenanft/projetoFinal>. Acesso em 30/06/2016.

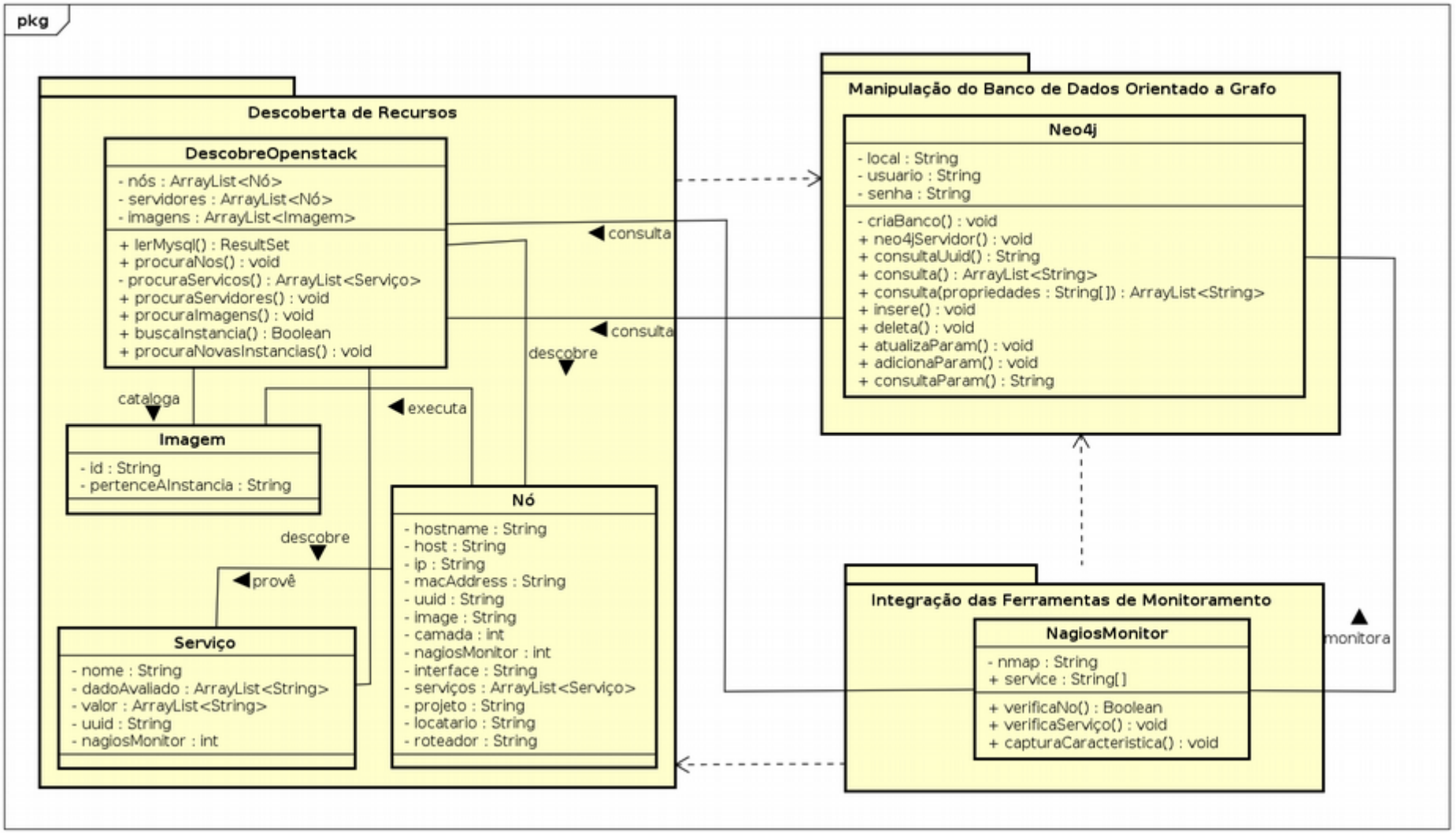

Figura 13: Diagrama de Classes.

4.1.1 Descoberta de Recursos

A IaaS escolhida como plataforma modelo para a aplicação foi o *Openstack*. Desse modo, a ferramenta precisa executar no nó controlador do mesmo, onde se encontra a base de dados com as informações necessárias. A execução da ferramenta proposta a solucionar o problema inicia com o método *procuraServidores*, capturando as informações dos nós de computação, onde as instâncias são executadas. A função faz uma consulta de leitura ao banco de dados *Mysql* do *Openstack*, ilustrada através da Figura 16, através do método *lerMysql*. Para essa e demais consultas ao *Mysql*, utiliza-se o JDBC do *Java*, como mostra a Figura 14. Assim, as informações coletadas dos nós de computação, são: nome da máquina (*hostname*), endereço IP (*ip*), a representação do contexto do nó (*camada*) e UUID gerado através do *Java* (*uuid*).

```
//Propriedades da Conexão
Properties connectionProps = new Properties();
connectionProps.put("user", user);
connectionProps.put("password", passwd);

// Carrega o Driver MySQL
Class.forName("com.mysql.jdbc.Driver");

// Configuração da Conexão com o Banco de Dados
connect = DriverManager.getConnection(
        "jdbc:" + "mysql" + "://" +
        "localhost" +
        ":" + "3306" + "/" + database,
        connectionProps);

// Statements Permitem Emitir Consultas SQL ao Banco de Dados
statement = connect.createStatement();

// Resultset Captam o Resultado da Consulta SQL
resultSet = statement.executeQuery(selectSql);
return resultSet;
```

Figura 14: Trecho de Código do Método *lerMysql*.

Antes que seja realizada a captura de informação das instâncias, é feito um catálogo das imagens utilizadas pelas mesmas, através do método *procuraImagens*.

Tais informações estão armazenadas na base de dados do módulo *Glance* do *Openstack* e a consulta também é ilustrada através da Figura 16. Assim, um objeto da classe *Imagem* possui: o UUID da imagem e o UUID da instância que está executando a mesma.

No *Openstack*, as imagens podem ser disponibilizadas por terceiros ou podem ser *snapshots* de outras instâncias. Assim, só há duas maneiras de realizar a migração de VMs: através de *snapshots* e através de uma cópia via *script*. Uma vez que ao ser fazer o *snapshot* o UUID da imagem é o mesmo UUID da instância clonada e na cópia via *script* mantém-se o UUID da VM, a função *procuraNos* é capaz de identificar a migração de máquinas. Isso porque é feita uma comparação entre o UUID da instância com os armazenados na classe *Imagem*. Essa identificação é um importante passo na gerência de um ambiente virtual.

A função *procuraNos* é responsável por desencadear o processo de identificação das instâncias. As informações coletadas que dizem respeito ao nó, são: nome do nó de computação (*host*), nome da máquina (*hostname*), endereço IP (*ip*), endereço MAC (*macAddress*), a interface de rede associada ao IP (*interface*), UUID da instância (*uuid*), UUID da imagem (*image*), a representação do contexto do nó (camada), o máximo de tentativas do *Nagios* (*nagiosMonitor*), a lista de serviços (*serviços*), o UUID do projeto (*projeto*), UUID da interface de rede do roteador (*roteador*) e o UUID do locatário (*locatário*). É importante salientar que as todos UUIDs utilizados pelas instâncias são providos pelo *Openstack*. A Figura 14 mostra como a coleta de nós é feita no banco de dados do *Openstack*.

Ainda na descoberta de instâncias do *Openstack*, também são coletados os serviços associados a cada uma, através do método *procuraServico*. Assim, a ferramenta utilizada para realizar essa coleta é o *NMap* e sua chamada a partir do *Java* é exemplificada pela Figura 15. Sendo que as opções *T3* e *sV* do *NMap* indicam: que a velocidade do rastreio deve ser média e que modelo e versão do serviço devem ser coletados, respectivamente. Com isso, inicialmente, um objeto da classe *Serviço* pode possuir os seguintes atributos: nome do serviço (*nome*), o UUID do serviço (*uuid*) e o máximo de tentativas do *Nagios* (*nagiosMonitor*). Contudo, as aplicações de monitoramento podem coletar mais informações respectivas a um serviço, esses dados são armazenados nos seguintes atributos: *dadoAvaliado*, que

mantém o nome das características avaliadas e *valor*, que armazena os valores dessas características. É importante salientar que o UUID do serviço também é gerado através do *Java*.

```
//Execução da Biblioteca NMap4j do Java
Nmap4j nmap4j = new Nmap4j ( "/usr" );
nmap4j.addFlags ( "-T3 -sV" );
nmap4j.includeHosts( ip );
nmap4j.execute() ;
```

Figura 15: Execução do *Nmap* Através de Código *Java*.

As funções *procuraNovasInstancias* e *buscaInstancia* são utilizadas pelas ferramentas de monitoramento. A primeira faz uma consulta ao banco de dados do *Openstack* em busca de novas instâncias e a segunda busca uma instância em específico. Suas consultas também são demostradas através da Figura 16.

```
//Consulta SQL para a Coleta das Informações dos Nós de Computação
String selectSql = "select host, host_ip from compute_nodes;";

//Consulta SQL para a Coleta das Informações dos Nós Feita pelo Método procuraNos
String selectSql = "select host, hostname, uuid, image_ref, project_id,"+
            "user_id, network_info from instances i "+
            "join instance_info_caches ic on i.uuid = ic.instance_uuid "+
            "where vm_state='active';";

//Consulta SQL para a Coleta das Informações das Imagens pelo Método procuraImagens
String selectSql = "select image_id, value from glance.image_properties "+
              "where name='instance_uuid';";

//Consulta SQL para a Coleta do UUID das Instancias pelo Método buscaInstancia
String selectSql = "select uuid from instances "+
              "where uuid='"+uuid+"' AND vm_state='active';";

//Consulta SQL para a Coleta do UUID das Instancias ativas pelo Método procuraNovasInstancias
String selectSql = "select uuid from instances "+
              "where vm_state='active';";
```

Figura 16: Consultas Realizadas pelos Métodos em DescobreOpenstack.

Deste modo, a classe *DescobreOpenstack* mantêm: a lista de nós ativos com seus respectivos serviços, o servidor e as imagens utilizadas pelas instâncias ativas. A Figura 17 ilustra os métodos utilizados pela classe.

```
private static void procuraServidores()
throws Exception

public static ResultSet lerMysql(String selectSql, String user, String passwd, String database)
throws ClassNotFoundException, SQLException

public static void procuraImagens()
throws ClassNotFoundException, SQLException

public static void procuraNos()
throws Exception

private static ArrayList<Service> procuraServicos(String ip)
throws NMapInitializationException, NMapExecutionException

public static void procuraNovasIntancias()
throws Exception

public static boolean buscaIntancia(String uuid)
throws Exception
```

Figura 17: Assinatura dos Métodos da Classe *DescobreOpenstack*.

4.1.2 Manipulação do Banco de Dados Orientado a Grafo

O banco de dados orientado a grafo utilizado neste trabalho foi o *Neo4j*. Uma vez que as informações do banco de dados do *Openstack* foram coletadas através das classes do pacote *DescobreOpenstack*, a classe *Neo4j*, que contêm como parâmetros: o local do banco de dados, o usuário e a senha, faz consultas a fim de escrever e ler a topologia de recursos na base de dados do *Neo4j*. Desse modo, a função *criaBanco* é a principal responsável pela construção do grafo. Essa função chama o método *insere* com o propósito de criar: o nó de computação, as instâncias e seus respectivos serviços. A função *insere* é responsável por realizar quaisquer consultas de escrita ou leitura. Para todas as funções descritas nesta subseção, a conectividade com o *Neo4j* é realizada através do *driver* JDBC. A Figura 18 exemplifica este processo.

```
// Configuração da Conexão com o Banco de Dados
Neo4jConnection con=null;
try {
	con = new Driver().connect(local, props);
} catch (SQLException e1) {
// TODO Auto-generated catch block
	e1.printStackTrace();
}

// Resultsets Captam o Resultado da Consulta Cypher
ResultSet rs;
try {
	rs = con.createStatement().executeQuery(query);
} catch (SQLException e) {
	// TODO Auto-generated catch block
	e.printStackTrace();
}
```

Figura 18: Conexão JDBC com o *Neo4j*.

A Figura 19 demonstra as consultas *Cypher* de escrita que criam a topologia de recursos descrita na seção 3.2.2.

```
//Criação do Nó de Computação (server)
insere("CREATE (a:Compute{Name:'"+server.getHostname()+
      "',IP:'"+server.getIp()+
      "',Layer:"+server.getCamada()+
      ",uuid:'"+server.getUuid()+"'})");

//Criação das Instâncias (virtual)
insere("CREATE (a:Instance{Name:'"+virtual.getHostname()+
        "',IP:'"+virtual.getIp()+"',MAC:'"+virtual.getMac()+
        "',Interface:'"+virtual.getIface()+
        "',Layer:"+virtual.getCamada()+",uuid:'"+virtual.getUuid()+
        "',Project:'"+virtual.getProject()+
        "',Tenant:'"+virtual.getTenant()+
        "',Router:'"+virtual.getRouter()+"'})");

//Criação de Serviços (service)
insere("CREATE (a:Service{Name:'"+service.getName()+
      "',uuid:'"+service.getUuid()+"'})");

//Cria Relacionamento entre Nó de Computação e Instância
insere("MATCH (a:Compute),(b:Instance) WHERE a.uuid='"+server.getUuid()+
"' AND b.uuid='"+virtual.getUuid()+"' CREATE (a)-[r:Hosts]->(b)");

//Cria Relacionamento entre Instância e Serviço
insere("MATCH (a:Instance),(b:Service) WHERE a.uuid='"+virtual.getUuid()+
"' AND b.uuid='"+service.getUuid()+"'CREATE (a)-[r:Provides]->(b)");

//Criação do nó de Estado do Serviço
insere("create(n:ServiceStats)");
```

Figura 19: Consultas *Cypher* da Classe *Neo4j*.

Por fim, o método *neo4jServidor* é responsável por ligar, desligar e restartar o banco. As demais funções dessa classe servem para apoiar a integração das ferramentas de monitoramento. Assim, *consulta*, *consulta* com sobrecarga, *consultaParam* e *consultaUuid* realizam consultas de leitura, fazendo uso da expressão *RETURN* do *Neo4j*. Enquanto que os métodos *deleta*, *adiciona* e *adiciona* com sobrecarga executam consultas de escrita, explorando as expressões *SET* e *DELETE* do *Neo4j*.

A assinatura dos métodos da classe é exemplificada através da Figura 20.

```
private void criaBanco()

public static void neo4jServidor(String status)

public static String consultaUuid(String mode)

public static ArrayList <String> consulta (String mode, String property)

public static ArrayList <String> consulta (String mode, String property,
                    String propertyParam, String valueParam)

public static String consultaParam(String mode, String property)

public static void insere(String query)

public static void deleta(String mode, String property, String value)

public static void adiciona(String uuid, String property,
                String oldValue, String newValue)

public static void adiciona(String label, String property, String value)
```

Figura 20: Assinatura dos Métodos da Classe *Neo4j*.

4.1.3 Integração das Ferramentas de Monitoramento

Com a topologia de recursos criada em um banco de dados orientado a grafos, é possível orquestrar a gerência, selecionando:

- Ferramentas de monitoramento;
- Elementos da topologia de recursos, a saber: nó de computação, instâncias e serviços;
- Atributos dos elementos.

Foi desenvolvido um módulo para ser integrado à ferramenta, que utiliza os *plugins* do Nagios. Assim, a classe *NagiosMonitor* é responsável por implementar isso, tendo como atributos os caminhos desses *plugins* dentro do sistema de arquivos.

Os métodos *verificaNo* e *verificaServiço*, ilustrados na Figura 21, são responsáveis por monitorar instâncias e serviços, respectivamente. Ambas as funções realizam consultas ao *Neo4j* e decrementam a variável *nagiosMonitor*, caso o retorno do *plugin* seja diferente de *Ok* ou *Warning*. Assim, se o atributo estiver com

o valor igual a zero, o recurso é consultado na base dados a fim de verificar qualquer mudança de estado em seus atributos. Caso a instância esteja indicando estar desativada, a mesma é retirada do grafo. Se houver mudança nos atributos, os dados são atualizados e o monitoramento prossegue.

Para os serviços, caso tudo esteja como antes, é emitido um evento de tratamento a fim de tentar o restabelecimento do respectivo serviço. Se o erro persistir, o elemento é retirado do grafo, uma vez que está indisponível para a aplicação de monitoramento.

```
private void verificaServico (String nodeIp)

private boolean verificaNo (String nodeIp) throws Exception

private void capturaCaracteristica (String servico)

//Adiciona Características Específicas de um Serviço
adiciona(uuid, caracteristica, value);
```

Figura 21: Assinatura dos Métodos da Classe *NagiosMonitor*.

No *Openstack*, o acesso às instâncias é feito, principalmente, pelo SSH através de um par de chaves criptográficas. Por exemplo: *ssh -i mypair.pem ubuntu@10.0.0.1*. Assim, dispensa-se a configuração de módulos do *Nagios* como o NRPE, pois pode-se desenvolver *plugins* para monitorar recursos básicos, como: disco, armazenamento e processamento, com o uso do SSH. Desse modo, o método *capturaCaracteristica* coleta algumas características de cada serviço e adiciona ao grafo do *Neo4j*.

4.2 Ambiente de Testes

O ambiente foi configurado numa máquina com as seguintes capacidades técnicas e recursos:

- Processador: Intel(R) Core(TM) i5-2410M CPU @ 2.30GHz.
- Memória (RAM): *4GB; DDR3*.

- Disco Rígido: Capacidade - *750GB; Western Digital;* Modelo - *WDC WD7500BPVT-8* .
- Sistema Operacional: *Ubuntu 14.04*.
- Configurações da rede: foi utilizada a configuração da rede local.
- *Neo4j 2.2.5*.
- *Openstack Kilo*.
- *Nagios Core 3*.

Para que a IaaS fosse configurada, foi utilizado o *script Devstack*, disponível por desenvolvedores do *Openstack* (OPENSTACK). A Figura 22 mostra as principais configurações de rede utilizadas neste trabalho. Como pode-se observar, a rede interna (*10.0.0.0/24*) é provida pela interface virtual configurada através do *Open vSwitch*. Desse modo o nó de computação possuirá o endereço *10.0.0.1* na rede privada e *10.0.10.1* na rede pública, internet.

```
###NETWORK
HOST_IP=10.0.0.1
SERVICE_HOST=10.0.0.1
MYSQL_HOST=10.0.0.1
RABBIT_HOST=10.0.0.1
GLANCE_HOSTPORT=10.0.0.1:9292

# Do not use Nova-Network
disable_service n-net

# Enable Neutron
ENABLED_SERVICES+=,q-svc,q-dhcp,q-meta,q-agt,q-l3

## Opções do Neutron
Q_USE_SECGROUP=True
FLOATING_RANGE="10.0.10.0/24"
FIXED_RANGE="10.0.0.0/24"
Q_FLOATING_ALLOCATION_POOL=start=10.0.10.40,end=10.0.10.50
PUBLIC_NETWORK_GATEWAY="10.0.10.1"
Q_L3_ENABLED=True
PUBLIC_INTERFACE=p5p1

# Open vSwitch provider networking configuration
Q_USE_PROVIDERNET_FOR_PUBLIC=True
OVS_PHYSICAL_BRIDGE=br-ex
PUBLIC_BRIDGE=br-ex
OVS_BRIDGE_MAPPINGS=public:br-ex
```

Figura 22: Configurações de Rede Utilizadas no *Devstack*.

Com o fim de realizar testes foi levantado um cenário com três máquinas virtuais, cada uma com seus respectivos serviços de rede. A Figura 23 mostra o cenário a partir do painel de controle do *Horizon*.

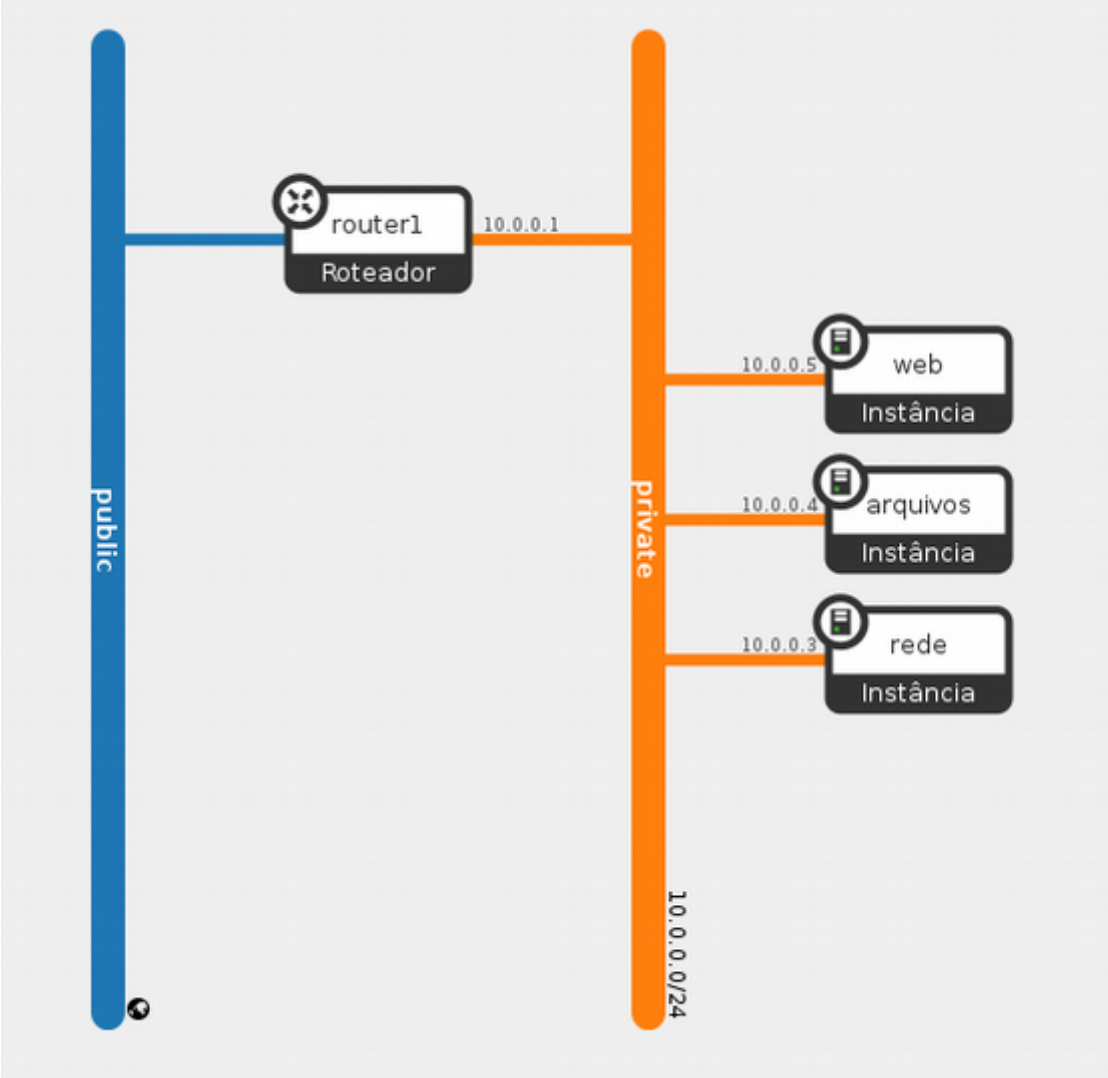

Figura 23: Topologia de Recursos Representada pelo *Horizon* do *Openstack*.

Para poder manipular os módulos deste trabalho, foi necessário desenvolver a classe *Orquestrador*, como mostra a Figura 24. A classe *Orquestrador* executa os módulos *DescobreOpenstack* e *Neo4j*, construindo a topologia de recursos. Por fim, o *orquestrador* seleciona todas as instâncias do grafo para monitorar. A cada laço de repetição são verificadas novas instâncias e a qualquer momento o usuário pode cancelar a execução do programa através da combinação de teclas '*ctrl*' e 'c'.

```
public class Orquestrador {
    public static void main(String[] args) throws Exception {
    System.out.println("Selecione a IaaS...");
    System.out.println("No momento, somente OPENSTACK é suportado");
    new DescobreOpenstack();
    Neo4j.neo4jServidor("start");
    new Neo4j();
    System.out.println ("NAGIOS MONITOR");
    //A execução do programa termina com um ctrl+c
    createShutDownHook();
        while(true){
            DescobreOpenstack.procuraNovasIntancias();
            try {
            //Executa nagios monitor
            ArrayList <String> ips;
            ips= Neo4j.consulta("INSTANCE", "IP");
            for(String ipNeo4j: ips)
                new NagiosMonitor(ipNeo4j);
            Thread.sleep(30000); //1000 milisegundos = 1 segundo
            }catch(InterruptedException ex) {
               Thread.currentThread().interrupt();
            }
        }
    }

    private static void createShutDownHook()
    {
      Runtime.getRuntime().addShutdownHook(new Thread(new Runnable()
      {
        @Override
        public void run()
        {  System.out.println();
           System.out.println("Obrigado por utilizar a aplicação");
           System.out.println("Saindo...");}
      }));
    }
}
```

Figura 24: Modelo de Classe que Realiza a Orquestração.

Os testes realizados, ilustrados pelas Figuras 25, 26 e 27, envolveram:

- Desligar a instância - Com isso, observa-se a execução do *NagiosMonitor*, retirando instâncias.
- Religar Instância - Aqui é observado a atuação da função *procuraNovasInstancias* inserindo ou reinserindo instâncias no grafo.
- Desativar e reativar serviço - Assim, observa-se a execução do *NagiosMonitor* removendo serviços caso não respondam ou incluindo caso estejam

novamente ativos.

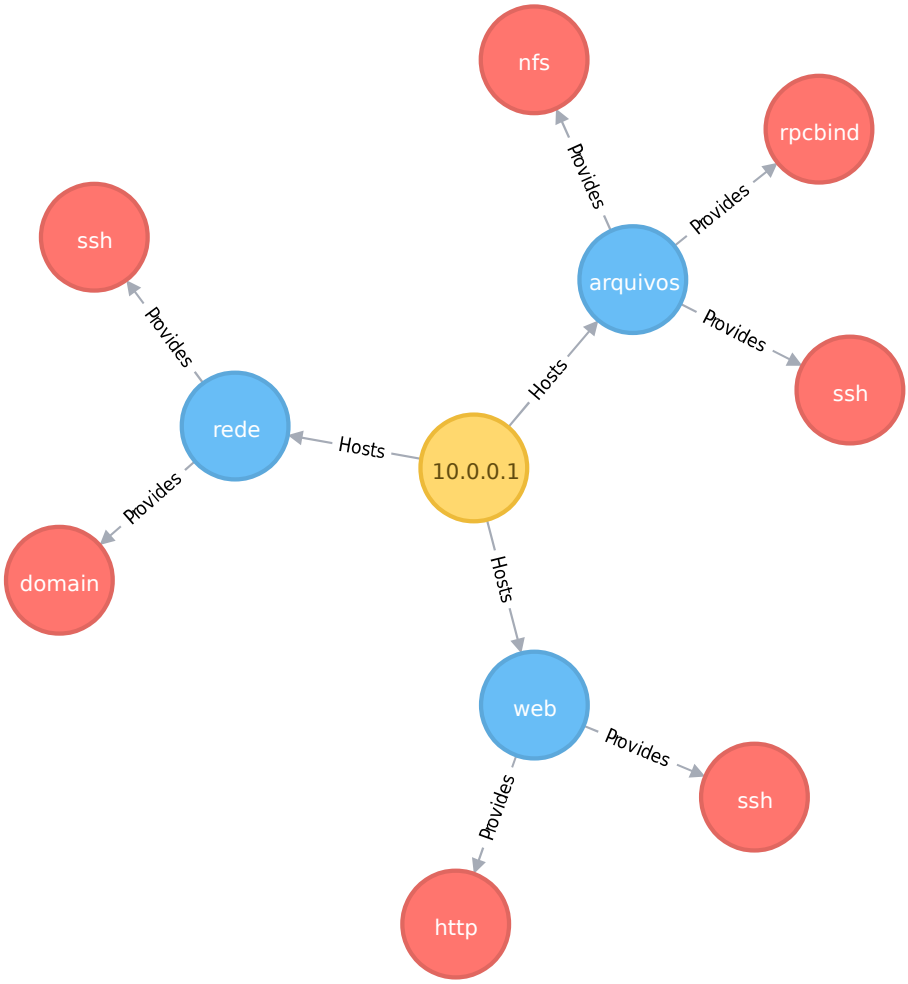

Figura 25: Consulta ao *Neo4j* Retornando a Topologia de Recursos Completa.

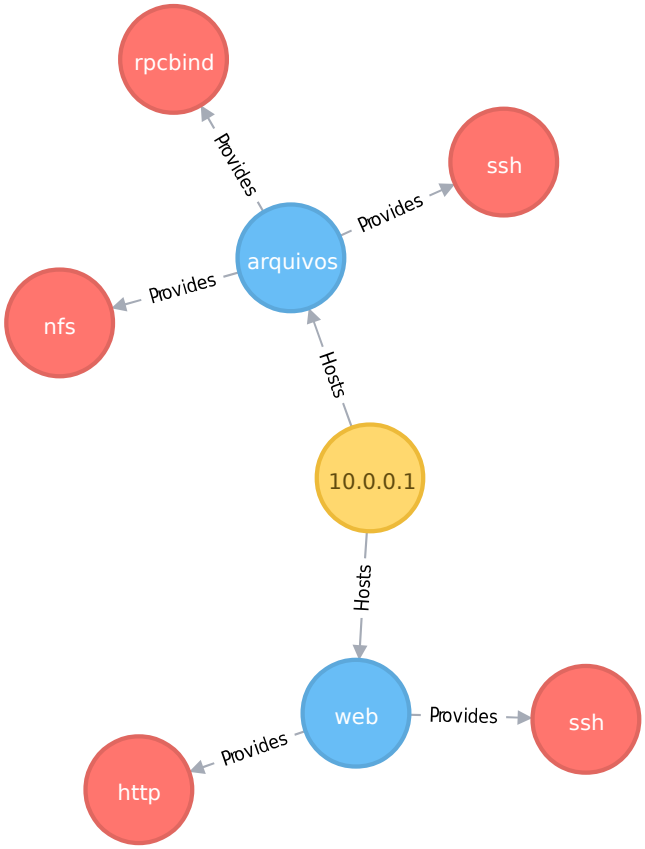

Figura 26: Consulta ao *Neo4j* Indicando Instância Ausente.

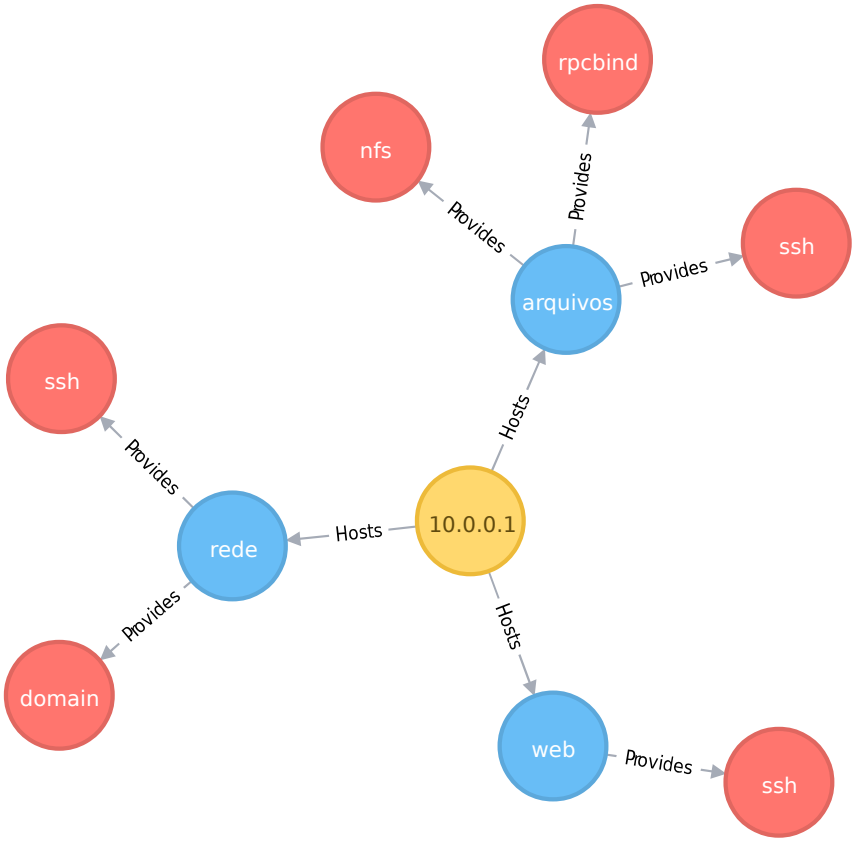

Figura 27: Consulta ao *Neo4j* Indicando Serviço Ausente.

O teste utilizou apenas a orquestração de dois dos três casos de uso descritos. A Figura 28 detalha de forma mais completa as possibilidades de utilização do *middleware*. Em um primeiro momento, pode-se acoplar, além do *Nagios*, o módulo *Ceilometer*, desenvolvendo uma classe *Java*. Num segundo momento, pode-se selecionar apenas alguns serviços de uma dada instância com um dado UUID. E, por fim, pode-se optar por monitorar apenas algumas características desejáveis de um dado serviço ou instância.

```
//Exemplo de Utilização do Sistema de Gerência Ceilometer
    while(true){
        DescobreOpenstack.procuraNovasIntancias();
        try {//Executa ceilometer monitor
        ArrayList <String> ips;
        ips= Neo4j.consulta("INSTANCE", "IP");
        for(String ipNeo4j: ips)
            new CeilometerMonitor(ipNeo4j);
        Thread.sleep(30000); //1000 milisegundos = 1 segundo
        }catch(InterruptedException ex) {
            Thread.currentThread().interrupt();
        }
    }

//Exemplo para um Possível Monitoramento de Serviços de uma Instância com Respectivo uuid
    while(true){
        DescobreOpenstack.procuraNovasIntancias();
        try {ArrayList <String> nomeServico;
        nomeServico = Neo4j.consulta("SERVICE", "Name", "uuid", uuid);
        for(String nomes: nomeServico)
            new NagiosMonitor(nomes);
        Thread.sleep(30000); //1000 milisegundos = 1 segundo
        }catch(InterruptedException ex) {
            Thread.currentThread().interrupt(); }
        }

//Exemplo para um Possível Monitoramento de Propriedades dos Serviços
    while(true){
        DescobreOpenstack.procuraNovasIntancias();
        try {ArrayList <String>  propriedadesServico;
        propriedadesServico = Neo4j.consultaParam("SERVICE", propriedade);
        for(String propriedades: propriedadesServico)
            new Monitor(propriedades);
        Thread.sleep(30000); //1000 milisegundos = 1 segundo
        }catch(InterruptedException ex) {
            Thread.currentThread().interrupt(); }
        }
```

Figura 28: Exemplo de Implementação dos Casos de Uso do Sistema.

4.3 Dificuldades na Solução do Problema

Todo o trabalho envolve tecnologias recentes cujo a documentação fornecida pode confundir novos usuários. Isso ocorre tanto no *Neo4j* como no *Openstack* e ambos os sites oficiais dispõem o conteúdo de forma pouco amigável, com informações de difícil acesso. A exceção é o *Nagios*, programa consolidado no mercado há mais de quinze anos.

5 CONSIDERAÇÕES FINAIS

5.1 Conclusões

As ferramentas de monitoramento em um ambiente em nuvem devem cumprir certos requisitos descritos em (ACETO et al., 2013), citados no capítulo introdutório deste trabalho. Através do desenvolvimento da solução proposta, pode-se aprimorar, principalmente, a compreensibilidade e a autonomicidade das ferramentas de monitoramento em relação à infraestrutura. Por meio do trabalho realizado foi possível capturar os elementos básicos da topologia de recursos, como: máquina física, máquinas virtuais e respectivos serviços. Com isso, foi concebível estabelecer as dependências entre os recursos e realizar inferências a partir do grafo semântico.

O uso de um banco de dados orientado a grafo ajudou na compreensão de uma topologia de recursos, dispondo-a de forma integrada e provendo uma interoperabilidade semântica. Com a integração de dados elaborada pelo *middleware* desenvolvido no trabalho, as ferramentas podem atuar de forma mais precisa. Isso porque, aumenta-se a possibilidade de fazer inferências mais completas a partir da visão ampla da nuvem provida pelo banco de dados orientado a grafo. Consequentemente, o esforço da ferramenta em realizar a descoberta dos recursos da rede é evitado e, como resultado, foi possível adaptar a ferramenta de monitoramento *Nagios* de forma relativamente simples. Qualquer ferramenta de monitoramento que desempenha o trabalho de gestão da rede através de *scripts* e *plugins* pode ser integrada de forma semelhante à abordagem feita com o Nagios, descrita na seção 4.1.3. E, caso o banco de dados orientado escolhido seja o *Neo4j*, a integração pode ser realizada através de qualquer linguagem suportada pelo mesmo.

Em Neo4j (NEO4J GRAPH DATABASE) e (ROBINSON; WEBBER; EIFREM, 2015) é constatada a melhor performance dos banco de dados orientado a grafo em relação à outras formas de persistência de dados como os bancos relacionais e orientados a documento. Desse modo, como o *Neo4j* pode ser acessado de diversas máquinas, pode-se evitar possíveis gargalos, distribuindo o monitoramento através

de outras máquinas na rede. Além disso, o *Neo4j* possibilita o acesso via HTTPS, aumentando a segurança. Portanto, a abordagem proposta por este trabalho provê um passo a mais para as pesquisas recentes no contexto da gerência na computação em nuvem.

5.2 Limitações e Perspectivas Futuras

Apesar dos resultados alcançados e do cumprimento dos requisitos da orquestração, algumas práticas podem ser tomadas para aumentar a eficiência da orquestração da gerência de recursos. Tanto a descoberta de recursos como a integração de ferramentas de monitoramento, podem ser feitas de forma mais precisa, como apontam diversos estudos levantados em (ACETO et al., 2013). Em (CALERO et al., 2015), por exemplo, onde é feita a integração do *Nagios* com o *Openstack,* foi apresentada uma abordagem eficiente de descoberta e gerência de recursos, em um ambiente com mais de 3500 máquinas virtuais. Consequentemente, para comprovar a eficiência da proposta deste trabalho, também é necessário realizar testes em grande escala.

Por fim, pode-se constatar que a detecção automática do tipo de IaaS; o detalhamento mais completo de toda a infraestrutura física; a utilização de uma variável indicando o estado do recurso e, assim, não remover o mesmo do grafo, apenas alterando o respectivo estado; a utilização de um mecanismo auxiliar para a descoberta de recursos, como feita em (CALERO et al., 2015), e aprimoramento nas restrições ao banco de dados orientado a grafo, para que locatários não tenham acesso à infraestrutura física, por exemplo, são possíveis contribuições futuras que este trabalho pode receber.

6 REFERÊNCIAS BIBLIOGRÁFICAS

WANG, L. et al. Cloud computing: a perspective study. **New Generation Computing**, v. 28, n. 2, p. 137-146, 2010.

ACETO, G. et al. Cloud monitoring: A survey. **Computer Networks**, v. 57, n. 9, p. 2093-2115, 2013.

WANG, C. et al. A flexible architecture integrating monitoring and analytics for managing large-scale data centers. In: **Proceedings of the 8th ACM international conference on Autonomic computing**. ACM, p.141-150, 2011.

MIAN, R.; MARTIN, P.; VAZQUEZ-POLETTI, J. L. Provisioning data analytic workloads in a cloud. **Future Generation Computer Systems**, v. 29, n. 6, p. 1452-1458, 2013.

HASSELMEYER, P.; D'HEUREUSE, N. Towards holistic multi-tenant monitoring for virtual data centers. In: **Network Operations and Management Symposium Workshops (NOMS Wksps), 2010 IEEE/IFIP**. IEEE, p. 350-356, 2010.

KATSAROS, G.; KÜBERT, R.; GALLIZO, G. Building a service-oriented monitoring framework with rest and nagios. In: **Services Computing (SCC), 2011 IEEE International Conference on**. IEEE, p. 426-431, 2011.

SHIREY, R. W. Internet security glossary, version 2. 2007.

CLAYMAN, S. et al. Monitoring, aggregation and filtering for efficient management of virtual networks. In: **Proceedings of the 7th International Conference on Network and Services Management**. International Federation for Information Processing, p. 234-240, 2011.

FALBO, R. A. et al. Ontologias e ambientes de desenvolvimento de software semânticos. **Jornadas Iberoamericanas de Ingeniería del Software e Ingeniería del Conocimiento**, 2004.

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.

MELL, P.; GRANCE, T. The NIST definition of cloud computing., p. 20-23, 2011.

OPENSTACK. **Openstack**. Disponível em : <http://docs.openstack.org/>. Acesso em: 14 jun. 2016.

OPENVSWITCH. **Open vSwitch**. Disponível em : <http://openvswitch.org/>. Acesso em: 21 jun. 2016.

ANGLES, R.; GUTIERREZ, C. Survey of graph database models. **ACM Computing Surveys (CSUR)**, v. 40, n. 1, p. 1-39, 2008.

ROBINSON, I.; WEBBER, J.; EIFREM, E. **Graph Databases: New Opportunities for Connected Data**. " O'Reilly Media, Inc.", 2015.

BARTHÉLEMY, M.; CHOW, E.; ELIASSI-RAD, T. Knowledge Representation Issues in Semantic Graphs for Relationship Detection. In: **AAAI Spring Symposium: AI Technologies for Homeland Security**, p. 91-98, 2005.

NEO4J GRAPH DATABASE. **Neo4j**. Disponível em : <http://neo4j.com/docs>. Acesso em: 21 jun. 2016.

ELMROTH, E.; LARSSON, L. Interfaces for placement, migration, and monitoring of virtual machines in federated clouds. In: **2009 Eighth International Conference on Grid and Cooperative Computing**. IEEE, p. 253-260, 2009.

BARI, Md. F. et al. Data Center Network Virtualization: A Survey. In: **IEEE Communications Surveys & Tutorials**, v. 15 , n. 2 ,p.909-928, 2013.

NAGIOS CORE. **Nagios**. Disponível em : <https://assets.nagios.com/downloads/nagioscore/docs/nagioscore/4/en/toc.html>. Acesso em: 16 jun. 2016.

NMAP SECURITY SCANNER. **Nmap**. Disponível em : <https://nmap.org>. Acesso em: 17 jun. 2016.

APACHE CLOUDSTACK. **Apache CloudStack**. Disponível em : <http://cloudstack.apache.org/>. Acesso em: 14 jun. 2016.

OPENNEBULA. **OpenNebula**. Disponível em : <http://opennebula.org/>. Acesso em: 14 jun. 2016.

EUCALYPTUS. **Eucalyptus**. Disponível em : <http://www8.hp.com/us/en/cloud/helion-eucalyptus-overview.html>. Acesso em: 14 jun. 2016.

MONTEIRO, M. E. et al. An Ontology-based Approach to Improve SNMP Support for Autonomic Management. In: **10th International CNSM**, Rio de Janeiro, p.280-283, 2014.

CALERO, J. M. A. et al. MonPaaS: an adaptive monitoring platformas a service for cloud computing infrastructures and services. **IEEE Transactions on Services Computing**, v. 8, n. 1, p. 65-78, 2015.

KANG, J. et al. Software-Defined Infrastructure and the SAVI Testbed. **Lecture Notes Of The Institute For Computer Sciences, Social Informatics And Telecommunications Engineering,** p.3-13, 2014.