UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

Pedro Henrique Pereira Martins

PROPOSTA DE UM MODELO AUTÔNOMO PARA MONITORAMENTO DE AMBIENTES DE COMPUTAÇÃO EM NUVEM

Florianópolis

2016

Pedro Henrique Pereira Martins

PROPOSTA DE UM MODELO AUTÔNOMO PARA MONITORAMENTO DE AMBIENTES DE COMPUTAÇÃO EM NUVEM

Trabalho de conclusão de curso submetida ao Ciências da Computação para a obtenção do Grau de Bacharel em Ciências da Computação.
Orientador
Universidade Federal de Santa Catarina: Prof. Dr. Carlos Becker Westphall
Coorientador
Universidade Federal de Santa Catarina: Me. Rafael Weingärtner

Florianópolis

2016

Ficha de identificação da obra elaborada pelo autor através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

A ficha de identificação é elaborada pelo próprio autor

Maiores informações em:
http://portalbu.ufsc.br/ficha

Pedro Henrique Pereira Martins

PROPOSTA DE UM MODELO AUTÔNOMO PARA MONITORAMENTO DE AMBIENTES DE COMPUTAÇÃO EM NUVEM

Esta Trabalho de conclusão de curso foi julgada aprovada para a obtenção do Título de "Bacharel em Ciências da Computação", e aprovada em sua forma final pelo Ciências da Computação.

Florianópolis, 22 de julho 2016.

Prof. Dr. Carlos Becker Westphall
Coordenador
Universidade Federal de Santa Catarina

Banca Examinadora:

Prof. Dr. Carlos Becker Westphall
Orientador
Universidade Federal de Santa Catarina

Me. Rafael Weingärtner
Coorientador
Universidade Federal de Santa Catarina

Profa. Dra. Carla Merkle Westphall
Universidade Federal de Santa Catarina

RESUMO

No cenário atual as ferramentas de orquestração de ambientes de computação em nuvem como Apache CloudStack, OpenStack e outras não possuem um módulo nativo de monitoramento do ambiente. Para monitorar o ambiente de computação em nuvem, são utilizadas normalmente extensões destas ferramentas ou outras aplicações como Nagios, Zenoss e outras que satisfazem parcialmente a necessidade de monitoramento. Contudo, estas ferramentas são incompletas ou possuem outros problemas como dados incorretos, violam as máquinas virtuais dos usuários (injetando scripts ou aplicações nas máquinas virtuais) além de possuírem uma deficiência de autonomia. Logo, os administradores do ambiente precisam ficar monitorando constantemente o ambiente para conseguir configurar a ferramenta de monitoramento para se adequar ao ambiente monitorado. Esta proposta consiste em especificar e implementar uma plataforma autônoma para monitorar a infraestrutura de ambientes de computação em nuvem. Palavras-chave:

Palavras-chave: Computação em nuvem. Monitoramento autônomo. Apache CloudStack. Plataforma de monitoramento.

ABSTRACT

Cloud computing orchestration tools such as Apache CloudStack, OpenStack and others do not have a comprehensive native monitoring module. To monitor the cloud environment we normally use plugins or applications such as Nagios, Zenoss, and others that satisfy partially the monito- ring needs. However, they are incomplete or have problems such as accuracy, intrusiveness (injecting scripts or applications in the users virtual machine), and a lack of autonomy. Therefore system administrators have to constantly monitor and configure all of the aplications/plugins parameters to best fit the environment. The goal of this proposal is the specification and development of an autonomous monitoring platform for cloud computing infrastructure.

Keywords: Cloud computing. Autonomous monitoring. Apache CloudStack. Monitoring Platform.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

SUMÁRIO

1 INTRODUÇÃO

Computação em nuvem vem sendo cada vez mais utilizada devido ao seu modelo permitir um melhor aproveitamento dos recursos de hardware e software, acarretando em redução de custos, melhor eficiência energética, elasticidade, flexibilidade e disponibilidade de recursos por demanda (ARMBRUST et al., 2010) e (SINGH; SOOD; SHARMA, 2011).

Com o aumento da demanda por serviços prestados pela nuvem, há um constante crescimento da infraestrutura de modo a atender a demanda (SINGH; SOOD; SHARMA, 2011) e (WEINGÄRTNER; BRÄSCHER; WESTPHALL, 2015). O crescimento constante dos ambientes da nuvem acaba aumentando a complexidade de sua gerência tornando esta inviável de ser realizada por administradores sem auxilio de ferramentas apropriadas (WEINGÄRTNER; BRÄSCHER; WESTPHALL, 2015), (HALL, 2012) e (URGAONKAR; SHENOY; ROSCOE, 2002).

Kephart e Chess (2003) apresentam um modelo de computação autônoma como uma solução para lidar com grandes ambientes computacionais heterogêneos e complexos como ambientes de computação em nuvem. Adicionalmente Kephart e Chess (2003) definem um elemento autônomo constituído de quatro (4) fases, monitoramento, análise, planejamento e execução, onde cada fase depende da anterior. Cada fase é detalhada a seguir.

- Monitoramento: durante este processo são coletadas informações sobre os recursos alocados e utilizados. Este processo é realizado periodicamente de modo a criar um histórico de alocação e uso de recursos;

- Análise: após ter os dados coletados, são iniciados os processos de análise em busca de tendências de consumo e comportamento das cargas e problemas referentes a carga que competem por recursos no mesmo servidor físico;

- Planejamento: depois de identificar as tendências de comportamento das cargas do ambiente, e possíveis problemas de degradação de serviço, pode-se traçar ações de modo a evitar as degradações de serviço e viabilizar a otimização do ambiente;

- Execução: após ter traçado uma sequência de ações serem tomadas é necessário executar todas as ações que foram previamente

planejadas de modo a buscar atender aos objetivos do sistema de gerenciamento.

Os processos descritos anteriormente são executados constantemente, de modo que seja possível para o sistema de gerenciamento identificar o contexto em que se encontra e adaptar-se ao ambiente a ser gerenciado de modo a atender aos objetivos gerenciais.

Uma das necessidades para se efetuar o monitoramento do uso de recursos de ambientes de nuvem se dá para fins comerciais, como cobrança relativa a serviços prestados. O valor a ser cobrado pode ser acordado entre o usuário e o fornecedor de recursos da nuvem ou pode ser pré-definido pelo fornecedor (ACETO et al., 2012). Além disso, o monitoramento também pode ser usado para viabilizar técnicas de otimização e auxiliar numa melhor gerência dos ambientes de computação em nuvem. Para viabilizar tais otimizações é importante que o monitoramento forneça dados precisos e em tempo hábil sem afetar o funcionamento da nuvem (ACETO et al., 2012) e (VIRATANAPANU et al., 2010).

O monitoramento em tempo real da nuvem é desafiador. Deve-se levar em conta o grande número de serviços e usuários, que torna a nuvem um ambiente com um grande volume de informação para ser coletada e analisada (SHAO et al., 2010) e (CLAYMAN; GALIS; MAMATAS, 2010).

Outro fator que contribui para a complexidade de monitorar os ambientes de nuvem é a heterogeneidade dos seus componentes. Ambientes de nuvem são constituídos de diferentes clusters de servidores, cada um com seus respectivos processadores, memória, placa de rede, periféricos e virtualizadores. Deste modo, é necessário implementar meios para monitorar cada um desses elementos distintos (SHAO et al., 2010). Os elementos que compõem o ambiente de computação em nuvem, podem estar dispersos geograficamente, o que gera a necessidade de ter um sistema de monitoramento distribuído, com nodos colhendo informações em cada elemento e enviando para um repositório de dados.

O processo de monitoramento pode vir a sobrecarregar os recursos da nuvem se não for devidamente implantado, sendo este um dos principais desafios para o monitoramento dos recursos (SHAO et al., 2010) e (CLAYMAN; GALIS; MAMATAS, 2010). Além disso, após implantado o sistema de monitoramento, qualquer modificação necessária deve ser feita sem a necessidade de desligar ou reiniciar o ambiente (CLAYMAN; GALIS; MAMATAS, 2010).

Problemas resultantes de um monitoramento mal sucedido são:

- A queda de desempenho da nuvem devido a sobrecargas de processamento com monitoramento;

- Ações de gerência mal sucedidas devido a dados inconsistentes advindos do processo de monitoramento;

- Perda de dados e/ou obtenção de dados corrompidos durante o processo de atualização dos componentes de monitoramento.

Atualmente as ferramentas de orquestração de computação em nuvem (levantadas durante o desenvolvimento deste trabalho) em sua maioria, não realizam o monitoramento dos recursos utilizados ou realizam mas não armazenam de modo histórico os dados monitorados. O que estas ferramentas costumam adotar é o monitoramento momentâneo do ambiente para indicar aos administradores o estado corrente deste, sem a criação de um histórico de uso dos recursos.

Hoje em dia existem ferramentas que realizam o monitoramento e podem ser acopladas a ferramentas de orquestração. Porém são de difícil instalação e dependentes de APIs (*Appication Programming Interface*) de dos orquestradores que não são completas pois não permitem o monitoramento de uma série de recursos físicos. Algumas ferramentas de monitoramento costumam apresentar dados inconsistentes com o ambiente, além de ser mais uma camada de software para gerenciar.

O Zenoss (ZENOSS, 2015) é um exemplo de ferramenta de monitoramento que apresentam alguns problemas. No caso do Zenoss, e sua extensão para monitoramento do Apache CloudStack (APACHE, 2015b), o CloudStack ZenPack (CLOUDSTACKZENPACK, 2015), existe um problema no monitoramento de uso de memória do servidor físico. A ferramenta faz uma requisição do uso de memória para o Xen (INC., 2015) e projeta o retorno como se fosse o uso total de memória das máquinas virtuais alocadas no servidor. O problema é que esta requisição do Xen, retorna apenas o uso de memória relativo ao *dom0*. Deste modo o Zenoss acaba fornecendo dados de apenas uma máquina virtual específica e não do servidor monitorado como um todo, o que pode levar a decisões inconsistentes no momento de gerenciamento e otimização dos ambientes.

É necessário fazer uso de boas técnicas de manipulação de dados devido ao grande volume de informações que podem ser monitoradas em ambientes de nuvem. Adicionalmente a ferramenta de monitoramento necessita ser o menos invasiva possível, ou seja, ser invisível para os usuários de ambientes de computação em nuvem. Além disso,

deve-se tratar a precisão das informações capturadas, os dados fornecidos devem condizer com o ambiente monitorado (ACETO et al., 2012). Quanto mais rápidas as informações sobre o estado do ambiente forem disponibilizadas, mais eficientemente se conseguirá administrá-lo.

1.1 MOTIVAÇÃO

O monitoramento é de fundamental importância para ambientes de nuvem, pois proporciona um melhor entendimento do estado em que o ambiente se encontra, do estado das aplicações que estão hospedadas, do uso/disponibilidade de recursos físicos do ambiente e também para determinar se os acordos de prestação de serviço estão sendo devidamente cumpridos.

Como apresentado em (ACETO et al., 2013), existe uma carência de autonomia nas ferramentas de monitoramento atuais. Assim, toda vez que novos recursos são inseridos no ambiente, deve-se informar às ferramentas que tais recursos devem ser monitorados, quando o ideal era a ferramenta detectar e passar a monitorar os recursos autonomamente.

Outro ponto sobre as ferramentas atuais, é que a proposta delas em sua maioria, é monitorar desde a camada de aplicação até o nível de infraestrutura. Com isso, as ferramentas de monitoramento acabam se tornando complexas, com uma instalação complicada e consumindo mais recursos do ambiente do que se fossem específicas para apenas um modelo de serviço de nuvem.

Motivado pela carência de ferramentas de monitoramento pouco intrusivas, autônomas, com foco apenas na infraestrutura de ambientes de computação em nuvem, compatível com as ferramentas de orquestração atuais e de fácil instalação. Surgiu o interesse no projeto de uma abordagem que supra a necessidade de monitoramento da infraestrutura da nuvem. Dessa forma ferramentas como Apache CloudStack, podem fornecer todo o suporte para o monitoramento da infraestrutura do ambiente de nuvem sem interferência nos serviços por ela já fornecidos de modo autônomo e, sem configurações adicionais e dependência de administradores dos ambientes de nuvem.

1.2 OBJETIVOS

Esta seção irá tratar dos objetivos que este trabalho pretende alcançar.

1.2.1 Objetivo geral

Este trabalho busca analisar o tema de monitoramento de ambientes de computação em nuvem e elementos autônomos. Com isso, será proposto um modelo de monitoramento autônomo para ambientes de computação em nuvem que fornecem infraestrutura como serviço.

1.2.2 Objetivos específicos

Os objetivos específicos deste trabalho são:

- Levantar trabalhos relacionados e fazer um estudo sobre monitoramento de ambientes de computação em nuvem;
- Definir um modelo para realização de monitoramento autônomo de um ambiente de nuvem que forneça infraestrutura como serviço;
- Criar um *framework* a partir do modelo definido para viabilizar a implantação em ferramentas de orquestração de nuvem;
- Verificar o impacto que o *framework* pode causar no ambiente de nuvem monitorado.

1.3 ORGANIZAÇÃO DO TEXTO

O conteúdo deste trabalho é disposto nos seguintes capítulos:

1. Introdução: apresentação do o trabalho, motivação da escolha do tema, definição dos objetivos e organização do texto;
2. Fundamentação Teórica: proporciona um conhecimento básico sobre conceitos e tecnologias utilizados para o desenvolvimento do trabalho;
3. Trabalhos relacionados: apresenta trabalhos cuja proposta seja semelhante à proposta apresentada;

4. Proposta: é apresentado o problema e uma solução com base nos trabalhos relacionados, Bibliografia científica e ferramentas existentes;

5. Implementação: capítulo que descreve como a proposta foi implementada;

6. Conclusão: encerra o trabalho apresentando as contribuições, resultados finais e levantamento de trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são apresentados os conceitos e tecnologias utilizadas para o desenvolvimento deste trabalho.

2.1 COMPUTAÇÃO EM NUVEM

A computação em nuvem é um modelo ubíquo, conveniente e de acesso compartilhado a recursos computacionais como servidores, armazenamento, aplicações e redes (MELL; GRANCE, 2011). Estes recursos compartilhados podem ser rapidamente provisionados e liberados com um esforço mínimo de gestão ou interação com o provedor de serviços (MELL; GRANCE, 2011).

2.1.1 Características

Em (MELL; GRANCE, 2011) são definidas como características básicas de um ambiente de computação em nuvem:

1. Acesso aos recursos sob demanda - o usuário pode escolher quais serviços/componentes ele irá usar dentro do que é disponibilizado pela nuvem sem a necessidade de interação com administradores;

2. Disponibilidade - ambientes de computação em nuvem devem estar sempre disponíveis para o usuário pela internet independente da plataforma de acesso (*tablet*, *smartphone*, *desktop* e outros);

3. Recursos compartilhados - os recursos físicos da nuvem devem poder atender diferentes usuários simultaneamente;

4. Elasticidade - conforme um usuário solicita mais recursos, a nuvem deve alocar mais recursos para este usuário evitando a degradação dos serviços prestados, assim como se um usuário deixar de usar recursos, a nuvem pode remover recursos desnecessários;

5. Monitoramento de serviços - a nuvem deve ser capaz de monitorar o uso de recursos por cada serviço, proporcionando transparência aos fornecedores e usuários do serviço.

2.1.2 Modelos de serviço

Mell e Grance (2011) descrevem os modelos de serviço disponibilizados pela nuvem como sendo:

1. *Software as a Service* (SaaS) : o provedor de nuvem fornece uma aplicação final via rede para usuários. O usuário não tem controle algum sobre a infraestrutura que hospeda a aplicação ou da plataforma que a aplicação usa, ele apenas consegue usar a aplicação;

2. *Platform as a Service* (PaaS) : são disponibilizadas diferentes plataformas para o usuário desenvolver e hospedar suas próprias aplicações. Neste nível o usuário tem controle total das aplicações, e pode configurar por completo a plataforma. O usuário não possui acesso a recursos do sistema operacional que hospeda a plataforma;

3. *Infrastructure as a Service* (IaaS) : a nuvem fornece toda a infraestrutura para o usuário, como processador, disco, memória e rede. Neste nível o usuário tem uma máquina virtual pronta para uso, ele define quais plataformas ele irá usar e quais aplicações ele irá executar em cada plataforma, porém o usuário não tem acesso e controle dos componentes físicos do ambiente.

   Cada modelo de nuvem descrito visa um público diferente com diferentes propósitos, o software como serviço visa mais usuários de aplicações. O modelo de plataforma como serviço procura atender os desenvolvedores que usarão as plataformas para hospedar suas aplicações. Em contrapartida, o modelo de fornecimento de infraestrutura como serviço tem como foco empresas que desenvolvem serviços e os hospedam na infraestrutura de ambientes de computação em nuvem para minimizar seus custos com infraestrutura. Este trabalho tem como foco ambientes de nuvem computacional que atuam fornecendo infraestrutura como serviço.

2.2 VIRTUALIZAÇÃO

A virtualização é um processo que foi primeiramente desenvolvido pela IBM com intuito de particionar um *mainframe* com alta capacidade computacional (SAHOO; MOHAPATRA; LATH, 2010).

O processo de virtualização consiste na existência de um virtualizador que gerencia sistemas operacionais convidados e intermedeia toda a comunicação entre hardware e esses sistemas (MENASCÉ, 2005). Sistemas operacionais convidados são executados sobre o virtualizador como se fossem aplicações em um sistema operacional qualquer. Portanto os sistemas operacionais convidados, não possuem acesso direto aos dispositivos físicos.

O uso da virtualização possibilita que os mesmos componentes de hardware suportem diferentes sistemas operacionais como sendo diversas unidades lógicas. Estas unidades lógicas em que o hardware é dividido são nomeadas de *Virtual Machine* (VM), em português, máquina virtual. Os sistemas operacionais e aplicações dentro das máquinas virtuais detectam esses ambientes como reais. Ao final do processo de virtualização, para o usuário final, ter uma máquina virtual ou uma máquina física não irá mudar sua experiência de uso do computador (SAHOO; MOHAPATRA; LATH, 2010).

2.2.1 Vantagens e desvantagens da virtualização

A virtualização permite um melhor aproveitamento dos recursos de hardware disponíveis. Permite a alteração de peças de hardware sem a necessidade de parar o funcionamento das VMs por longos períodos, bastando migrar a VM para outro servidor físico. O custo alugar uma máquina virtual em um servidor de computação em nuvem é menor que manter uma máquina física equivalente. (SAHOO; MOHAPATRA; LATH, 2010).

Como problemas temos uma sobrecarga de processamento para gerenciar a inúmera quantidade de máquinas virtuais. Além disso o desenvolvimento de virtualizadores não é algo simples e é de grande importância ter redundância de hardwares pois se ocorrer defeito em um componente, todas as VMs vinculadas a tal componente serão penalizadas (POPEK; GOLDBERG, 1974) .

2.2.2 Tipos de virtualização

Nesta seção, serão discutidos os tipos de virtualizações segundo (BEN et al., 2010), e suas principais características.

- Virtualização por contêiner: este tipo de virtualização precisa de um sistema operacional base. Neste caso, o virtualizador irá criar

instâncias virtuais do sistema operacional base, isoladas logicamente. Estas instâncias são nada mais que ponteiros para os endereços físicos das rotinas do sistema operacional base. Isto gera um problema que não existe nos outros tipos de virtualização, que é o fato de todas as aplicações das instâncias dividirem os mesmos endereços físicos para suas chamadas de sistema, com isso, se o sistema operacional base for sucetível a algum ataque, uma aplicação maliciosa pode danificar o sistema operacional base, e com isso, prejudicar todas as outras instâncias virtuais desse sistema. Esse tipo de virtualização consome menos memória que as demais, pois usa apenas uma instância física de sistema operacional. Neste tipo de virtualização, os clientes só tem a possibilidade de criar VMs exclusivamente do mesmo tipo do sistema operacional base;

- Virtualização do tipo 1: este tipo de virtualização executa diretamente sobre o hardware, neste caso o virtualizador irá gerar abstrações de hardware completas para poder hospedar um sistema operacional qualquer. Isto cria um consumo extra de memória, pois cada instância irá ter seu próprio sistema operacional em memória. Este tipo de virtualização, possui a melhor performance de uso do processador dentre os tipos aqui apresentados. Neste tipo de virtualização, os clientes podem escolher seus próprios sistemas operacionais para serem instaladas nas VMs;

- Virtualização do tipo 2: esta virtualização assim como a virtualização por contêiner, precisa de um sistema operacional como base. Porém nesse caso, o virtualizador irá emular um hardware que será usado pelas instâncias virtuais criadas. Isso possibilita que as instâncias tenham seus próprios sistemas operacionais independentemente de qual é o sistema operacional base. O virtualizador não tem acesso direto ao hardware, podendo assim ser um virtualizador mais simples que da virtualização do tipo 1, já que é o sistema operacional base que irá tratar da comunicação com o mesmo. Esse tipo de virtualização é um meio termo entre a virtualização do tipo 1 e por contêiner.

Cada tipo de virtualização possui vantagens e desvantagens. O tipo 1, é o mais complexo entre os tipos de virtualização e o com menor penalidade de processamento. O tipo 2 usa um sistema operacional nativo como base para o funcionamento do virtualizador, o que remove parte da complexidade do virtualizador, porém acaba prejudicando o

desempenho devido a camada de software extra. A virtualização por contêiner é o tipo com menor uso de memória, porém ela amarra os usuários ao sistema operacional base onde é executado o virtualizador.

2.2.3 Virtualizadores

O virtualizador é responsável pela gerência dos recursos de hardware e fornecimento desses recursos para as máquinas virtuais. O responsável de fato pela gerência, criação e destruição das máquinas virtuais, é o *Virtual Machine Monitor* (VMM) (VMWARE, 2007). Portanto, toda vez que uma VM é criada, antes, é criado um VMM, e este passa a gerenciar a VM, e destruí-la quando necessário. Logo, existe um VMM para cada VM.

O Xen (INC., 2015) é um virtualizador do tipo 1 de código aberto disponibilizado pela Citrix (SYSTEMS, 2015a), que da suporte as arquiteturas x86 (MAZEGEN, 2007), x86 64 (CORPORATION, 2015b), IA32 (CORPORATION, 2015a), IA64 (DOSHI, 2015) e PowerPC (IBM, 2015). As VMs instanciadas com esse virtualizador, são denominadas de domínio sem privilégios (DomU) (SYSTEMS, 2015c). Além disso, existe uma VM que cuida de todo o fluxo de entrada e saída de dados dos DomU, que é denominada de domínio de controle (Dom0) (SYSTEMS, 2015b). Além de intermediar o acesso a entrada e saída de dados dos DomU, o Dom0 também fornece uma série de comandos para destruir, criar ou modificar cada DomU. Esses comandos de gerência disponibilizados no Dom0, são nativos da biblioteca XL (SYSTEMS, 2015e), porém, são de difícil usabilidade. O Xapi (SYSTEMS, 2015d), é uma abstração para esses comandos que torna mais simples o uso dos recursos de criação de VMs, clusters e migração de VMs entre servidores de um mesmo cluster.

O KVM (LINUX-KVM, 2015) assim como o Xen, é um virtualizador do tipo 1 de código aberto. Suportado pela Red Hat (REDHAT, 2015). O KVM é uma extenção do kernel do linux que permite a criação e gerência de VMs. O KVM usa o QEMU (QEMU, 2015) para emular o hardware que as VMs instanciadas irão usar. Um dos diferenciais do KVM é o fato dele não se preocupar com a gestão de diferentes servidores presentes em cluster, não tendo assim, qualquer custo de processamento para gerência dos servidores em um cluster. Usar esta abordagem no controle dos clusters, torna o KVM dependente de uma ferramenta de orquestração que dê suporte a gerência de servidores dentro de um cluster.

O ESXi (VMWARE, 2015a) é um virtualizador de código fechado e licença gratuíta que foi desenvolvido pela VMware (VMWARE, 2015c). Tem suporte apenas para arquiteturas x86 de 32 e 64 bits. É um virtualizador do tipo 1. Este virtualizador é usado no vSphere que é uma ferramenta de criação de ambientes de nuvem criada pela VMware. O ESXi só permite a migração de VMs para diferentes servidores em sua versão comercial.

O LXC (CONTAINERS., 2015) é um virtualizador por contêiner gratuito de código aberto suportado pela Canonical (CANONICAL, 2015). O LXC da suporte para criação de instâncias de distribuições linux, logo este virtualizador suporta todas as arquiteturas suportadas pelo linux. Instâncias criadas pelo LXC não possuem privilégios de root. O LXC está no repositório de software de distribuições linux, a instalação dele é bem simples.

VirtualBox (ORACLE., 2015c) é um virtualizador de código aberto do tipo 2 mantido pela Oracle (INC., 2015). Este virtualizador pode ser instalado em distribuições Windows, Linux, Macintosh, Solaris e da suporte para instâncias de Windows (NT 4.0, 2000, XP, Server 2003, Vista, Windows 7, Windows 8), DOS/Windows 3.x, Linux (2.4, 2.6 e 3.x), Solaris e OpenSolaris, OS/2, e OpenBSD.

Tabela 1 – Comparação entre virtualizadores analizados

	Empresa Responsável	Arquiteturas Suportadas	Tipo de virtualização	Tipo de Licensa
Xen	Citrix Systems	x86, x86-64, IA 64, PowerPC	1	Código aberto
KVM	Red Hat	x86, x86-64, IA 64, PowerPC	1	Código aberto
ESXi	VMware	x86, x86-64	1	Gratuita/Comercial
LXC	Canonical	x86, x86-64, IA 64, PowerPC	Contêiner	Código aberto
VirtualBox	Oracle	x86, x86-64, IA 64	2	Código aberto

Dentre os virtualizadores analizados, será utilizado neste trabalho o Xen devido a ser um virtualizador de código aberto e suportado por uma comunidade amplamente ativa. O fato do Xen ter seu código aberto, possibilita que toda uma comunidade interaja e possa contribuir com melhorias tanto na documentação quanto em correções de *bugs* e evolução de suas funcionalidades.

2.3 ORQUESTRAÇÃO DE AMBIENTES DE COMPUTAÇÃO EM NUVEM

As ferramentas de orquestração na nuvem servem para abstrair do administrador toda a parte de interação com os virtualizadores, serviços de armazenamento, dispositivos de redes e outros. Deste modo,

permite-se que o administrador não precise de amplos conhecimentos nas tecnologias de virtualização, rede, armazenamento e outras utilizadas para se criar o ambiente de nuvem.

Essas abstrações proporcionadas pelas ferramentas de orquestração tornam possível que o administrador da nuvem possa delimitar recursos físicos para cada usuário facilmente. Assim como também possibilita que usuários sem conhecimento nenhum de computação possam criar suas próprias máquinas virtuais.

2.3.1 Ferramentas de orquestração

Seguindo a análise realizada no trabalho (BRäSCHER, 2015), foram verificadas as seguintes ferramentas de orquestração para uso no decorrer deste trabalho.

- Apache CloudStack: o Apache CloudStack (APACHE, 2015b) é uma ferramenta de orquestração de código aberto, suportada pela fundação Apache (APACHE, 2015a) e desenvolvida em java. O Apache CloudStack da suporte para virtualizadores como KVM, VM-ware , Xen e Hyper-V (MICROSOFT, 2015a). Esta ferramenta da suporte aos sistemas de arquivos *Internet Small Computer System Interface* (iSCSI) (MICROSOFT, 2015b) e *Network File System* (NFS) (MICROSOFT, 2013);

- Eucalyptus: o Eucalyptus (HP, 2015a), assim como o Apache CloudStack, é uma ferramenta de orquestração de ambientes de computação em nuvem. Suportada pela HP (HP, 2015b) e desenvolvida em java e C. O Eucalyptus, usa o Walrus como sistema de arquivo e tem suporte ao virtualizador KVM;

- OpenStack: O OpenStack (OPENSTACK, 2015), da suporte aos mesmos virtualizadores que o Apache CloudStack, é uma ferramenta de código aberto desenvolvida em Python. Suporta os sistemas de arquivo Swift e Cinder.

Tabela 2 – Comparação entre ferramentas de orquestração analisadas

	Linguagem Suportada	Sistema de Arquivos	Tipo de Licensa
CloudStack	Java	iSCSI , NFS	Código aberto
OpenStack	Python	Swift , Cinder	Código aberto
Eucalyptus	Java , C	Walrus	Código fechado

Dentre essas ferramentas de orquestração em nuvem, foi optado o uso do Apache CloudStack devida a uma série de motivos, tais como:

- Suportado por uma fundação e uma comunidade amplamente ativa;
- Possui uma documentação detalhada;
- É codificada em java que é uma linguagem de programação bem conhecida para futuras extensões;
- Tem suporte para sistemas de arquivos conhecidos;
- É uma ferramenta de código aberto, o que possibilita um melhor conhecimento da ferramenta e implantação de melhorias.

2.4 MONITORAMENTO DE AMBIENTES DE COMPUTAÇÃO EM NUVEM

Nesta seção serão abordadas as dificuldades relacionadas ao monitoramento na nuvem, sua importância, as características que o monitoramento deve possuir e algumas ferramentas que realizam monitoramento de ambientes de computação em nuvem.

2.4.1 Características de ferramentas de monitoramento para ambientes de computação em nuvem

Segundo Aceto et al. (2012) existem 11 características que uma ferramenta de monitoramento para ambientes de computação em nuvem deve possuir para ter um funcionamento correto e sem degradar os serviços fornecidos:

1. Escalabilidade: a ferramenta pode ser considerada escalável se ela consegue suportar uma larga quantidade de objetos a serem monitorados (CLAYMAN; GALIS; MAMATAS, 2010). Na nuvem, tal característica tem importância devido a grande gama de componentes que podem existir;
2. Elasticidade: para uma ferramenta ser considerada elástica, ela deve conseguir suportar mudanças ocorridas no ambiente monitorado, tais como novos objetos para serem monitorados ou objetos deixando de ser monitorados (CLAYMAN; GALIS; MAMATAS,

2010). Isto é de fundamental importância para ambientes de nuvem, pois o ambiente vive em contínua mudança com a adição e remoção de recursos dinamicamente;

3. Adaptabilidade: uma ferramenta de monitoramento é adaptável, se ela consegue se ajustar a mudanças de cargas impostas ao ambiente sem que este seja penalizado e mantendo o seu serviço de monitoramento ativo e funcionando corretamente (CLAYMAN; GALIS; MAMATAS, 2010). Para realizar o monitoramento do ambiente, a ferramenta consome recursos de rede, armazenamento e processamento. A ferramenta deve se ajustar para consumir menos recursos em ocasiões que o ambiente se encontra sobrecarregado;

4. Pontualidade: a ferramenta pode ser considerada pontual, se ela é capaz de relatar eventos que acontecem no ambiente, como perda repentina de um recurso, ou sobrecarga de outro, em tempo hábil para estes serem tratados (WANG et al., 2011);

5. Autonomia: para a ferramenta ser considerada autônoma, ela deve ser capaz de se gerenciar, conseguindo responder a mudanças repentinas do ambiente sem a necessidade de uma outra entidade a supervisionando. Ao mesmo tempo em que torna a complexidade do seu funcionamento discreta para usuários e gerentes do ambiente;

6. Abrangente: uma ferramenta de monitoramento é abrangente, se ela consegue monitorar diferentes tipos de recursos e capturar diferentes métricas destes recursos (HASSELMEYER; D'HEUREUSE, 2010). Isto é importante, já que ambientes de nuvem costumam ser heterogêneos;

7. Extensividade: a ferramenta pode ser considerada extensível, se ela suporta a adição de novas métricas para monitorar, sem a necessidade de modificações no código fonte da ferramenta.

8. Não intrusiva: para a ferramenta não ser considerada intrusiva, ela deve poder ser implantada no ambiente sem precisar de modificações significativas neste (KATSAROS; KUBERT; GALLIZO, 2011);

9. Confiabilidade: uma ferramenta de monitoramento é dita confiável, se mesmo com falhas em diversos componentes, ela ainda continua fornecendo seu serviço normalmente com os componentes que ainda lhe restam (LAPRIE, 2008);

10. Disponibilidade: a ferramenta de monitoramento deve estar disponível sempre que os seus serviços forem solicitados. Isto é importante para ambientes da nuvem, pois estes estão em constante atividade (SHIREY, 2007);

11. Precisão: a ferramenta é considerada precisa se as informações disponibilizadas pela ferramenta contêm valores que expressem o mais aproximadamente possível o estado em que o objeto monitorado se encontra no momento que a medida foi realizada.

Além das características levantadas por (ACETO et al., 2012), também são descritas algumas dificuldades na área de monitoramento da nuvem.

- Grande volume de informação: a nuvem possui diversos usuários, assim como componentes e um grande tráfego de rede. Com isso, algumas horas de monitoramento contínuo, podem gerar Tera Bytes de informações para serem armazenadas;

- Mudança constante do ambiente: devido a elasticidade de um ambiente de computação em nuvem, as métricas a serem monitoradas podem mudar a qualquer momento, pode haver a necessidade de monitorar um novo elemento que não estava previsto;

- Sensibilidade de ambientes carregados: em ambientes de computação em nuvem muito carregados, a presença do monitoramento pode acarretar em sobrecarga de processamento, e acabar comprometendo o funcionamento de todo o ambiente;

- Garantir autonomia da ferramenta: esta demanda a possível existência de um laço que irá receber informações do ambiente de tempos em tempos (a própria ferramenta que deve obter essas informações), processar essas informações e repassar as ações que cada nodo deve executar (no caso de uma ferramenta distribuída);

- Heterogeneidade do ambiente: ambiente de computação em nuvem possui inúmeros dispositivos com diferentes arquiteturas e tecnologias. Deve-se conseguir prover suporte de monitoramento para cada um desses dispositivos e ainda manter um bom isolamento entre eles;

- Rastreamento de elementos da nuvem: o ambiente de nuvem permite que seus recursos sejam migrados de um dispositivo físico para outro. Deve-se garantir que a ferramenta consiga detectar

a nova localização de um recurso migrado e continuar monitorando este recurso em sua nova localização, em vez de apenas acusar uma falta do recurso em um local e a detecção de um novo recurso em outro.

O monitoramento na nuvem é de grande importância tanto para os clientes quanto para os fornecedores de serviços de nuvem. O monitoramento é um meio para se garantir um melhor controle dos dispositivos de hardware e das aplicações que são executadas nos ambientes de nuvem (ACETO et al., 2012).

O constante monitoramento do ambiente, auxilia na garantia da qualidade do serviço prestado para os usuários do ambiente, como por exemplo: disponibilidade e tempo de resposta das aplicações hospedadas. Para os fornecedores de serviços, o monitoramento é um modo de mensurar os recursos utilizados pelo usuário e então poder ser cobrada uma taxa referente ao uso dos mesmos (ACETO et al., 2012). Para ambos os casos, o monitoramento auxilia na prevenção e até mesmo correção de desvios no termo de prestação de serviços acordado por usuário e provedor (ACETO et al., 2012).

2.4.2 Ferramentas de monitoramento

Nesta seção serão descritas ferramentas de monitoramento de código aberto que são comumente usadas para monitorar ambientes de computação em nuvem.

- Nagios: o Nagios (NAGIOS, 2015) suporta o monitoramento de máquinas virtuais e serviços de armazenamento (CARON et al., 2012). O Nagios é uma ferramenta de código aberto que possui inúmeras extensões. Existem extensões que possibilitam o Nagios monitorar sistemas como Eucalyptus, CloudStack(NAGIOS., 2015a) e OpenStack (NAGIOS., 2015b);

- Zenoss: O Zenoss (ZENOSS, 2015) é uma ferramenta opensource escrita em Java com suporte a diversos virtualizadores, como por exemplo, KVM, XenServer e VMware. Usuários do CloudStack, tem a possibilidade de usar o Zenoss ao instalar a extensão (CLOUDSTACKZENPACK, 2015), usuários de OpenStack podem usar a extensão (OPENSTACKZENPACK, 2015). Estas extensões ficam limitadas pelas funcionalidades disponibilizadas através das APIs das ferramentas de orquestração;

- Nimbus: a Nimbus (PROJECT, 2015) é uma ferramenta de criação e gerência de ambientes de computação em nuvem, Suporta virtualizadores como Xen ou KVM e da suporte para monitoramento do ambiente através do *Simple Network Management Protocol* (SNMP) (NET-SNMP, 2015). Esta ferramenta é voltada especialmente para projetos relacionados a comunidade científica. A Nimbus visa autonomia no monitoramento tanto de aplicações do ambiente quanto na infraestrutura como um todo, no caso, ela não suporta monitoramento de clusters, apesar de monitorar aplicações sendo executadas nas VMs.

- Zabbix: o Zabbix (ZABBIX, 2015) é uma ferramenta de monitoramento de infraestrutura e aplicações. Esta ferramenta da suporte para abordagens multi agente ou sem nenhum agente (centralizado). Esta ferramenta é compatível com vCenter (VMWARE, 2015b) permitindo monitoramento destes ambientes. Além de ser compatível com o vCenter, o Zabbix permite o monitoramento de qualquer dispositivo de hardware que suporte *Intelligent Platform Management Interface* (IPMI) (INTEL, 2015) e dispositivos que suportem SNMP, conseguindo capturar as mesmas métricas suportadas por ambos os protocolos IPMI e SNMP. Usuários de CloudStack podem usar o Zabbix através da extensão (ZABBIX-CLOUDSTACK, 2015) e usuários de OpenStack através da extensão (ZABBIXAGENTADOPTION, 2015)

- Icinga: o Icinga (ICINGA, 2015) é uma ferramenta que pode ser instalada em qualquer distribuição Linux. Nesta ferramenta, o usuário deve especificar o que deve ser monitorado antes de inicializar a ferramenta, ou seja, qualquer novo item a ser monitorado, irá requerer uma reinicialização da ferramenta. Assim como o Nagios, esta ferramenta promete uma fácil extensibilidade, possibilitando que os usuários criem plugins para expandir o funcionamento da ferramenta, além de suportar todos os plugins desenvolvidos para o Nagios. Esta ferramenta monitora serviços de rede e consumo de recursos de hardware, como CPU e memória. Ela possui uma extensão para usuário do OpenStack (ICINGAOPENSTACKGIT, 2015), usuários de CloudStack não tem suporte ainda para uso do Icinga e acabam adotando o Nagios como substituto já que ambas as ferramentas são similares.

Dentre as ferramentas analisadas, todas suportam monitoramento de recursos de hardware usando protocolos padrões de monitoramento

Tabela 3 – Características suportadas pelas ferramentas analisadas em (ACETO et al., 2013)

	Nagios	Zenoss	Nimbus	Zabbix	Icinga
Escalabilidade					
Elasticidade					
Adaptabilidade					
Pontualidade		X		X	
Autonomia			X		
Abrangencia					
Extensividade	X			X	X
Não intrusiva					
Confiabilidade				X	
Disponibilidade					
Precisão					

como SNMP e IPMI, além de usar estes protocolos, usam também APIs de ferramentas de orquestração. Uma característica em comum dessas ferramentas, é a abordagem de coletar os dados e armazená-los de modo histórico. Todas usam o mesmo conceito de ter uma entidade centralizada que de tempos em tempos, envia mensagens para os nodos que estão nos servidores físicos coletarem os dados e enviarem para o nó central.

Como apresentado na tabela 3, as ferramentas analisadas não dão suporte a inúmeras das características abortadas em (ACETO et al., 2012), isso acontece porque as ferramentas em si não interagem diretamente com os Virtualizadores dos ambientes de computação em nuvem e sim com APIs das ferramentas de orquestração. Essas APIs muitas vezes são incompletas e inconsistentes, ou seja, não disponibilizam alguns tipos de métricas e alguns dados coletados não são condizentes com o real estado dos elementos monitorados. Outro ponto que as ferramentas adotam é a inserção de agentes dentro das máquinas virtuais para coletar os dados de cada máquina virtual e então juntar esses dados para obter as métricas de todo o ambiente, o que torna a ferramenta intrusiva.

A proposta deste trabalho será de criar uma ferramenta que suporte as características apresentadas na Tabela 3 e seção 2.4.1. A ferramenta irá usar a API dos Virtualizadores para conseguir obter os dados dos ambientes em vez de usar APIs das ferramentas de orquestração. Outra abordagem diferente da ferramenta proposta neste

trabalho, será uma inversão do paradigma adotado pelas ferramentas analisadas, que é o fato de um nodo central fazer requisições para os nodos folhas coletarem os dados e retornarem estes para o nodo central.

A proposta deste trabalho consiste em os nodos folhas enviarem periodicamente os dados coletados para o nodo central sem o nodo central precisar requisitar, isso gera um menor fluxo de rede e torna a ferramenta menos impactante para o ambiente. Outro diferencial da proposta deste trabalho será a autonomia da ferramenta. A ferramenta irá se ajustar continuamente ao ambiente de computação em nuvem que ela está monitorando, deixando de monitorar recursos removidos ou passando a monitorar novos recursos.

A ferramenta de monitoramento proposta será uma extensão para o Apache CloudStack. O Apache CloudStack conhece todo o ambiente onde ele foi instalado, possibilitando que suas extensões também conheçam o ambiente. Com estas informações a ferramenta poderá a detectar mudanças no ambiente de computação em nuvem e então se auto ajustar para realizar o monitoramento do mesmo.

3 TRABALHOS RELACIONADOS

Este capitulo aborda sobre modelos de monitoramento de ambientes de computação em nuvem, pontos fortes e fracos de cada modelo, e por fim, uma comparação entre os modelos apresentados e o modelo proposto neste trabalho.

Em (KATSAROS et al., 2012) é proposto um modelo para monitorar todas as camadas (aplicação, plataforma e infraestrutura) de um ambiente de computação em nuvem. O modelo consiste de 6 componentes constituindo a ferramenta de monitoramento como um todo, cada componente é descrito a seguir:

- Framework de serviço de monitoramento: é o principal componente, responsável por orquestrar os outros componentes e disponibilizar os resultados do monitoramento para o usuário;
- Repositório central: responsável por organizar e guardar todos os dados do monitoramento;
- Serviço de monitoramento de infraestrutura: responsável por obter dados do uso de recursos da infraestrutura;
- Serviço de monitoramento de instancia: responsável por obter dados do uso de recursos de cada VM de um servidor através do coletor de dados e guardar no repositório central;
- Repositório local: responsável por armazenar os dados de monitoramento de uma única VM;
- Coletor de dados: coleta os dados de uso de recursos de uma única VM e guarda no repositório local.

As principais características abordadas no modelo de (KATSAROS et al., 2012) é a elasticidade e acurácia, o modelo permite reconfigurar todas as métricas de monitoramento sem a necessidade de reiniciar o sistema ou proporcionar algum impacto de desempenho para o ambiente. O principal problema encontrado é que cada instancia de máquina virtual irá gerar uma instancia de coletor de dados e repositório local, o que gera um uso extra de memória; é uma solução invasiva, pois insere software dentro da VM do usuário.

Andreozzi et al. (2005) propõem um modelo de monitoramento de serviços distribuídos. O modelo proposto é dividido em 5 módulos independentes para facilitar extensões e manutenção. É apresentado

um módulo de coleta de dados que usa sensores (o tipo de sensor ou como são implementados não é especificado) para medir tráfego de rede, desempenho do serviço monitorado, disponibilidade do mesmo e uso de recursos físicos. O segundo módulo é o responsável por analisar quais dados são mais interessantes para cada tipo de serviço, para tal fim foi usado o Globus MDS (GLOBUS, 2015) para fazer a análise dos dados capturados e classificá-los . O terceiro módulo armazena cada tipo de dados em uma base de dados relacional, no caso o postgreSQL (POSTGRESQL, 2015). O quarto módulo serve para informar quando algum serviço está sobrecarregado/inacessível. E o ultimo módulo é responsável pela apresentação dos dados através de uma página web.

Este modelo foi feito com a intenção de monitorar serviços web e informar aos usuários as condições de cada serviço. O banco de dados para persistência dos dados separa cada tipo de dado em tabelas diferentes para agilizar consultas. O diferencial deste modelo é o fato de conseguir peneirar os dados não relevantes para cada tipo de serviço e não armazenar tal dado, porem ele ainda continua monitorando tal tipo de dado.

O modelo apresentado por (ANDREOZZI et al., 2005) permite adições de novos módulos e métricas de monitoramento, sendo que novos módulos requerem a reinicialização do sistema e novas métricas não precisam de reinicialização. Usa banco de dados relacional para persistir os dados e persiste apenas dados considerados relevantes. A visualização dos dados pode ser feita usando um histórico de 1, 5 e 15 minutos, históricos superiores a esses intervalos não são descartados.

Shao et al. (2010) propõem um modelo de monitoramento baseado em agentes que opera em todas as camadas de computação em nuvem e tem como principal característica balancear a capacidade de monitoramento da ferramenta com o aumento ou redução da carga dos servidores, de modo a ajustar a coleta de dados baseado na carga do ambiente. A ferramenta realiza isso medindo a carga do ambiente e ajustando a frequência de obtenção dos dados, com o ambiente com pouca carga a frequência de coleta de dados aumenta e vice-versa.

Para realizar a coleta de dados de cada VM, são instanciados agentes (Aplicações Java), dentro de cada máquina virtual, ou seja, cada máquina virtual vai ter uma *Java Virtual Machine* (JVM) executando um agente para obter os dados da VM. Este agente por sua vez, utiliza uma biblioteca nativa escrita em C (HYPERIC, 2015).

O balanceamento da carga do ambiente com a coleta de dados da ferramenta ocorre de modo manual, usando uma ferramenta do Java 6 *hotswap* que permite trocar uma classe java por uma outra versão dessa

Tabela 4 – Comparação dos Trabalhos relacionados

	(KATSAROS et al., 2012)	(ANDREOZZI et al., 2005)	(SHAO et al., 2010)
Modo de guardar dados	Banco relacional	Banco relacional	Banco relacional
Mantém histórico de dados	Sim	Por no máximo 15 min	Sim
Filtro de métricas	Não	Sim	Não
Área de atuação	IaaS, PaaS e SaaS	Serviços distribuidos	IaaS, PaaS e SaaS
Modo de obtenção dos dados	Script injetado nas VMs	Sensores	Aplicação Java injetada nas VMs
Permite ajustar intervalo de exibição dos dados	Não	Sim	Não

mesma classe em tempo de execução, assim se torna possível alterar os agentes em tempo de execução de forma a possibilitar um melhor controle de consumo de recursos pela ferramenta. A persistência dos dados ocorre num banco de dados relacional onde não são separados tipos de dados como no trabalho anterior.

Como mostrado na Tabela 4 dentre os trabalhos analisados, todos optam por uma abordagem de banco de dados relacional para a persistência dos dados coletados, talvez uma abordagem não relacional seja mais adequada para esse tipo de problema pois bancos não relacionais tendem a se tornar mais rápidos que relacionais quando o volume de dados se torna grande (LI; MANOHARAN, 2013). Uma diferenca desses trabalhos em relação as ferramentas mostradas, é o fato que eles propõem um modelo em que os dados coletados por cada nodo são enviados para um repositório periodicamente em vez de uma entidade central solicitar que tais nodos enviem os dados, como é feito nas ferramentas analisadas. Além disso, um problema encontrado nos

modelos propostos é o fato de como eles obtém os dados das máquinas virtuais. Usando métodos invasivos de inserção de scripts ou aplicações dentro das máquinas virtuais dos usuários, além desses métodos serem invasivos, eles são pouco práticos, pois acabam ficando presos ao sistema operacional que o usuário escolher, um script para obter dados num Unix por exemplo, vai ser diferente do script de um Windows.

Além disso, como mostrado na Tabela 4 todas as propostas tendem a manter um histórico dos dados armazenados, porém não disponibilizam um meio de configurar por quanto tempo o usuário gostaria que os dados ficassem armazenados. Outra coisa apontada na Tabela 4 é o uso de um filtro de métricas pela proposta de (ANDREOZZI et al., 2005) que permite otimizar a coleta de dados do ambiente sem deixar de capturar a essência do comportamento do mesmo.

4 PROPOSTA

Conforme apresentado por Aceto et al. em (ACETO et al., 2012), o cenário de ferramentas de monitoramento para ambientes de computação em nuvem não atende as necessidades de gerência, seja por falta de precisão nos resultados do monitoramento, falta de extensibilidade da aplicação ou o fato de injetarem código nas máquinas virtuais dos usuários.

A proposta deste trabalho consiste na criação de uma plataforma de monitoramento de ambientes de computação em nuvem que possa atender às características de monitoramento elencadas em (ACETO et al., 2013) de modo autônomo e sem prejudicar o ambiente de computação em nuvem e seus usuários. Deste modo, será utilizada uma arquitetura orientada a serviços, onde cada serviço é responsável por garantir uma funcionalidade da ferramenta, sendo possível a adição de novos serviços em tempo de execução para complementar a plataforma de monitoramento de ambientes de computação em nuvem em nível de IaaS. Além disso, o uso de agentes para realizar a configuração autônoma do componente de monitoramento, permite que a plataforma se ajuste autonomamente ao comportamento do ambiente, possibilitando um melhor uso de recursos.

4.1 ARQUITETURA DA PLATAFORMA DE MONITORAMENTO

As ferramentas de orquestração de ambientes de computação em nuvem possuem 5 elementos básicos como descrito em (BRäSCHER, 2015)

- Gerente de identidades - é responsável por definir quais privilégios sobre cada recurso do ambiente cada usuário possui;
- Gerente de alocação - trata da alocação de máquinas virtuais para os usuários assim que estas são solicitadas;
- Componente virtualizador - intermedeia a comunicação entre diferentes tipos de virtualizadores e a ferramenta de orquestração;
- Componente de armazenamento - efetua o gerenciamento e compartilhamento de imagens das máquinas virtuais entre servidores do ambiente;

- Componente de rede - responsável por toda a configuração de rede do ambiente.

A plataforma proposta neste trabalho pode ser vista como um sexto elemento da ferramenta de orquestração que será a responsável por gerenciar o monitoramento das máquinas virtuais e servidores do ambiente de nuvem. Ao fim da implantação da plataforma de monitoramento, a distribuição dos elementos que compõe a ferramenta de orquestração ficará como ilustrado na Figura 1.

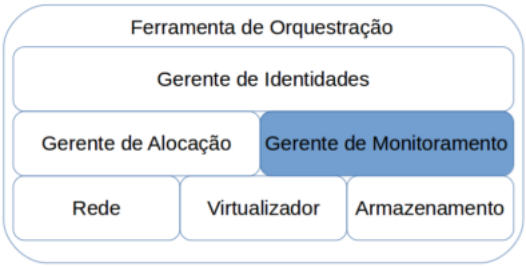

Figura 1 – Ferramenta de orquestração estendida com a plataforma autônoma de monitoramento

4.2 COMPONENTES DA PLATAFORMA DE MONITORAMENTO

Nesta sessão sã descritos os elementos existentes na plataforma de monitoramento, como eles são constituídos e qual papel de cada um. Ilustrada na Figura 2, a plataforma de monitoramento é composta de 3 componentes: o componente de monitoramento, de configuração e de armazenamento.

4.2.1 Componente de Monitoramento

Este componente é responsável por coletar os dados referentes ao consumo de recursos de cada máquina virtual e servidores do ambiente em que ele foi designado para atuar. Este componente é constituído de um ou mais serviços de captura de dados que podem monitorar as mesmas ou diferentes métricas como ilustrado na Figura 3 Cada serviço de captura poderá a coletar uma ou mais variáveis do ambiente, poderão haver múltiplas instâncias do mesmo serviço no ambiente, monitorando

o mesmo conjunto de métricas de servidores e máquinas virtuais diferentes, assim como também poderão haver diferentes serviços que monitoram conjuntos de métricas distintos.

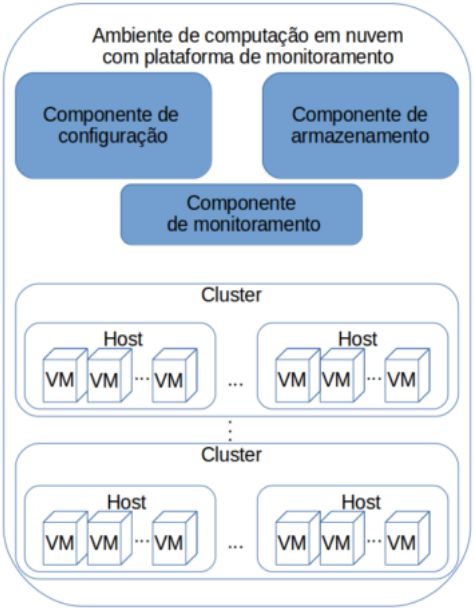

Figura 2 – Ambiente de computação em nuvem com a plataforma de monitoramento

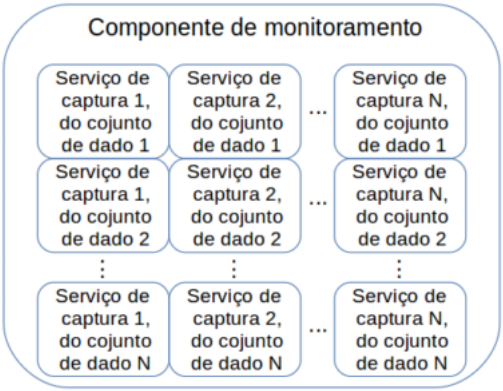

Figura 3 – Componente de monitoramento

Após ter obtido os dados, os serviços do componente de monitoramento realizam o envio dos mesmos para um repositório de armazenamento através do componente de armazenamento. Os dados obtidos serão organizados no formato *JavaScript Object Notation* (JSON) , um exemplo dos dados coletados pode ser encontrado nas Figuras 4 e 5. Como ilustrado nas Figuras 4 e 5, cada JSON contem 6 propriedades:

1. *resourceType* - o tipo de recurso coletado;

2. *resourceUsage* - o quanto deste tipo de recurso a máquina virtual está usando no momento da coleta;

3. *virtualMachineName* - o identificador da máquina virtual monitorada;

4. *timestamp* - a data em que ocorreu a coleta;

5. *host* - o servidor cuja máquina virtual está sendo executada;

6. *cluster* - o cluster onde o servidor da máquina virtual se encontra.

```
{"resourceType":"cpu", "resourceUsage":"13%",
    "virtualMachineName": "VM_11", "timestamp":
    "11/05/2011 17:44:32","host": "Servidor 1",
     "cluster": "HVM-Intel"}
```

Figura 4 – Exemplo de JSON gerado através de dados coletados relativos ao uso de cpu.

```
{"resourceType":"memoria", "resourceUsage":"22
    1mb", "virtualMachineName": "VM_11", "
    timestamp":"11/05/2011 17:44:32","host": "
    Servidor 1", "cluster": "HVM-Intel"}
```

Figura 5 – Exemplo de JSON gerado através de dados coletados relativos ao uso de memória.

4.2.1.1 Serviço de captura de dados

O serviço de captura de dados irá conter uma aplicação que através da API do virtualizador obtém as métricas de consumo ou alocação de recursos para as máquinas virtuais e desempenho dos servidores físicos como ilustrado na Figura 6. Deste modo, o serviço de captura de dados garante que não seja necessário a injeção de qualquer tipo de código nas máquinas virtuais dos usuários ao mesmo tempo em que caso algum novo dispositivo de hardware seja inserido no servidor, este possa ser monitorado por algum serviço de monitoramento existente no ambiente, pois esse dispositivo só poderá ser usado após ser reconhecido pelo virtualizador, que por sua vez irá disponibilizar o acesso aos recursos do dispositivo através da sua API.

O serviço de captura de dados deverá se cadastrar na plataforma de monitoramento disponibilizando as métricas que este suporta. O serviço de captura de dados também deverá verificar se o servidor que foi designado para ser monitorado está ativo (isto deverá ser feito toda vez que o serviço for coletar as métricas do servidor); se o servidor não estiver disponível, o serviço deverá remover esse servidor da sua

lista de servidores para monitorar. Além do cadastro, o serviço de captura de dados deverá disponibilizar os seguintes endpoints que irão dar forma a seguinte *Uniform Resource Locator* (URL) *"http://ip-serviço:8080/<endpoint>"*:

- *timeSpent* - retorna o tempo médio que o serviço leva para coletar os dados de cada servidor que ele é responsável;
- *hosts* - retorna todos os servidores monitorados pelo serviço;
- *addHost/host* - adiciona um servidor para ser monitorado pelo serviço;
- *removeHost* - remove um servidor qualquer da lista de servidores monitorados pelo serviço;
- *setDataRetrieveInterval/value* - altera o valor do intervalo entre coletas de dados;
- *getDataRetrieveInterval* - retorna o valor do intervalo entre coletas de dados.

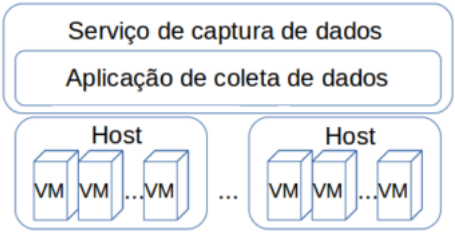

Figura 6 – Exemplo de um serviço de captura de dados

4.2.2 Componente de armazenamento Ocomponente

Ocomponente de armazenamento será constituído de uma aplicação e uma base de dados como como ilustrado na Figura 7. Este componente realiza o armazenamento e gerência dos dados oriundos do monitoramento através de uma aplicação que recebe requisições *Hypertext Transfer Protocol* (HTTP) contendo um JSON com as informações sobre uma máquina virtual ou servidor para persistir no banco de dados. Além disso, a aplicação disponibiliza métodos para consultar informações sobre o ambiente monitorado, como: recursos utilizados por

máquinas virtuais, recursos alocados e utilizados dos servidores físicos, percentual de recursos ociosos em um cluster e outros.

Ambientes de computação em nuvem possuem milhares de máquinas virtuais, servidores e outros componentes como dispositivos de rede e armazenamento. Deste modo, como modelo de banco de dados será utilizado uma base de dados não relacional orientada a documentos, visto que esses tipos de banco de dados possuem um desempenho melhor em leitura e escrita num grandes volumes de dados como apresentado em (LI; MANOHARAN, 2013). Para a modelagem do banco de dados não relacional foi criado um modelo *Enhanced Entity-Relantionship* (EER) ilustrado na Figura 8. O modelo definem 7 entidades:

1. *Host* - entidade que irá conter informações relativas a um servidor;
2. *VM* - entidade que irá armazenar informações de máquinas virtuais;
3. *HostResource* - esta entidade armazena dados relativos ao monitoramento de servidores;
4. *VMResource* - esta entidade irá armazenar os dados obtidos ao monitorar máquinas virtuais;
5. *ServiceRegistry* - entidade responsável por armazenar o registro de cada tipo de serviço de captura de dados da plataforma;
6. *HostActivity* - entidade que irá armazenar um histórico de atividades dos servidores, se foram ligados ou desligados e quando isto ocorreu;
7. *ServiceSupport* - esta entidade armazena as características dos serviços, os virtualizadores e métricas que eles suportam.

4.2.3 Componente de Configuração

O componente de configuração é composto por dois agentes apresentados na Figura 9. O agente de configuração determina a frequência de coleta de dados dos serviços de captura. Ele contém uma lista com todos os serviços disponíveis no ambiente e fará requisições periódicas para o serviço de armazenamento a fim de obter métricas do ambiente para poder definir melhores configurações de intervalo de monitoramento para cada serviço de captura de dados. O procedimento que o

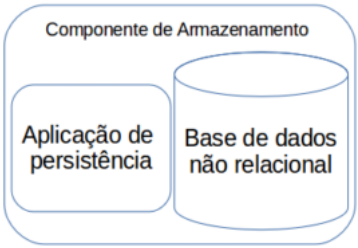

Figura 7 – Componente de armazenamento de dados

agente irá a usar para a melhoria da configuração dos serviços é descrito no Algoritmo 1.

O Algoritmo 1 tem como objetivo melhorar o intervalo de funcionamento de cada serviço de coleta de dados, fazendo uso da variação dos dados coletados por esses serviços. Esta variação consiste na relação entre os valores das métricas obtidos em diferentes períodos, se esta variação for grande, significa que as máquinas virtuais dos servidores monitorados por esse serviço estão sofrendo mais variações, deste modo seria mais interessante que o serviço passe a coletar os dados destas máquinas virtuais mais frequentemente para conseguir obter dados mais condizentes com seu estado. Do mesmo modo, se a variação for pequena, significa que as máquinas virtuais monitoradas por esse serviço estão com um comportamento estável. Assim, o serviço de captura pode levar mais tempo para realizar a coleta dos dados dessas máquinas virtuais, reduzindo os custos com processamento, coleta e armazenamento de dados.

Descrição de cada variável e função presentes no Algoritmo 1:

- *servicoArmazenamento*[Linha: 1] - uma referência para o serviço responsável por realizar a persistência e consulta de dados no repositório da plataforma de monitoramento;
- *servicosConhecidos*[Linhas: 1 e 4] - uma lista contendo todas as instâncias dos serviços de captura de dados existentes na plataforma de monitoramento;
- *servico*[Linhas: 4,5,6,11,12 e 13] - uma instância de um serviço de captura de dados qualquer;
- *listaJson1 e listaJson2* [Linhas: 5,6,7 e 8] - listas de JSONs que armazenam todos os JSONs referentes as máquinas virtuais monitoradas num determinado período;

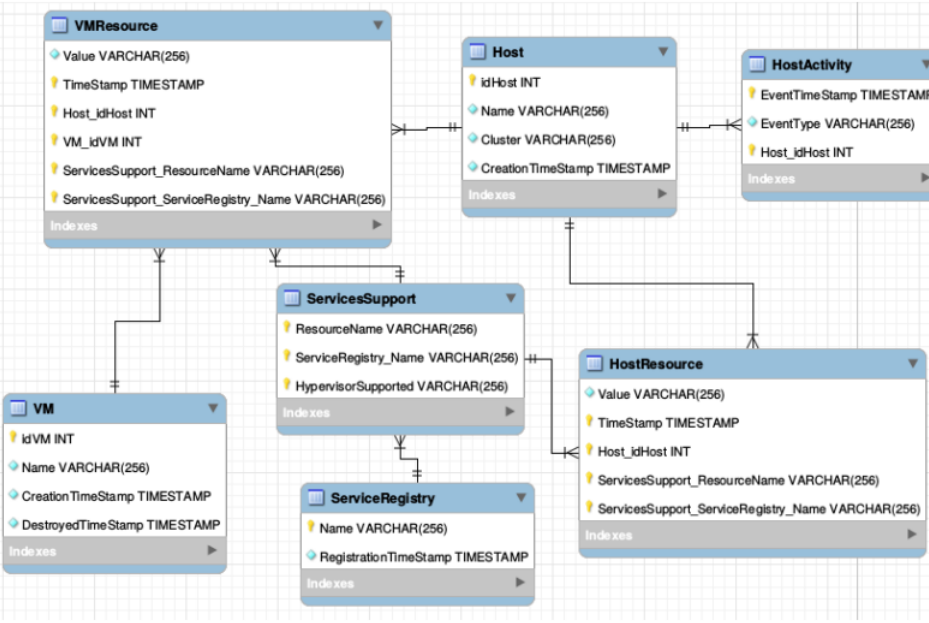

Figura 8 – Modelo do banco de dados

- *servidores*[Linhas: 5 e 6] - variável contendo os servidores em que o serviço atua;
- **dataAtual**()[Linhas: 2 e 3] - função que irá retornar o momento corrente;
- *período*[Linhas: 2 e 3] - variável que irá conter um tempo que será subtraído da data atual para gerar uma data no passado. O período pode ser configurado pelo administrador do sistema;
- **consultar**()[Linhas: 5 e 6] - método que irá retornar uma lista de JSON contendo todos os dados do banco de dados cujo atributo servidor esteja contido na lista de servidores e que as datas estejam dentro do intervalo de dados calculado com o uso do parâmetro período;
- **separarPorMetrica**()[Linhas: 7 e 8] - método que irá retornar uma lista de lista de JSON onde cada elemento é uma sublista

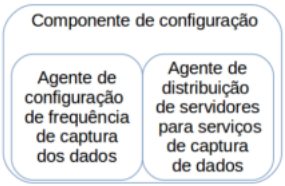

Figura 9 – Componente de configuração dos serviços de monitoramento

da lista de JSONs passada por parâmetro para o método, os elementos dessa sublista terão todos a mesma métrica e não poderão haver sublistas distintas contendo a mesma métrica;

- **obterMedias**()[Linhas: 9 e 10] - método que retorna um conjunto onde cada elemento é a média aritmética de todos as valores de uma métrica contida na lista de valores de métricas passada por parâmetro;
- *mediasDeMetricas1*[Linhas: 9 e 12] - array que contem a média dos valores de cada tipo de métrica coletada num passado próximo;
- *mediasDeMetricas2*[Linhas: 10 e 12] - array que contem a média dos valores de cada tipo de métrica coletadas num momento presente;
- **obterFrequencia**()[Linha: 11] - método que retorna a frequência com que um serviço de captura de dados está coletando os dados;
- **coefFreq**[Linhas: 12 e 13] - valor ao qual será multiplicada a frequência de coleta de um serviço de captura de dados para tornar o serviço mais eficiente;
- **coeficienteDeFrequencia**()[Linha: 12] - método que calcula o coeficiente de frequência utilizando como base a Função 4.1;
- **frequênciaInicial**()[Linha: 13] - método que retorna a frequência inicialmente definida no serviço de captura de dados;
- **nMetricas**()[Linha: 12] - método que retorna o número de métricas capturadas por um serviço de captura de dados;
- **definirFrequenciaDeCapturaDeDados**()[Linha: 13] - altera o valor da frequência de captura de dados do serviço de captura de dados usando a Fórmula 4.3 com os parâmetros passados.

Algorithm 1 otimizarIntervaloExecuçãoServiçosMonitoramento

1: **Entrada:** ServicoArmazenamento arm,ServicosConhecidos
2: *períodoAtual*←{**dataAtual**, **dataAtual** - *período*}
3: *períodoPassado*←{**dataAtual** - 2∗*período*, **dataAtual** - 4∗*período*}
4: **for all** *servico s* **em** *conhecidos* **do**
5: *listaJson1*←*arm*.**consultar**(s.**servidores**,*períodoPassado*)
6: *listaJson2*←*arm*.**consultar**(s.**servidores**,*períodoAtual*)
7: *listaDeListaJSON1*←**separarPorMetrica**(*listaJson1*)
8: *listaDeListaJSON2*←**separarPorMetrica**(*listaJson2*)
9: *mediasDeMetricas1*←**obterMedias**(*listaDeListaJSON1*)
10: *mediasDeMetricas2*←**obterMedias**(*listaDeListaJSON2*)
11: *frequenciaDeColeta*←*s*.**obterFrequencia**()
12: *coefFreq*←**coeficienteDeFrequencia**(*mediasDeMetricas1*,*mediasDeMetricas2*, *s*.**nMetricas**())
13: *s*.**definirFrequenciaDeCapturaDeDados**(*s*.***frequênciaInicial()***, *coefFreq*)
14: **end for**

Para realizar o ajuste da frequência foram definidas três Fórmulas 4.1, 4.2 e 4.3.

A Fórmula 4.1 receberá três valores, sendo eles:

- A *e* B - vetores de médias de métricas obtidas em diferentes períodos;
- h - número de métricas monitoradas pelo serviço;
- i - índice de iterações, valor que inicia em um (1) e sofrerá acréscimo de uma (1) unidade a cada iteração do produtório, podendo atingir um valor máximo de h.

A Fórmula 4.1 é constituída de um produtório sobre a Fórmula 4.2, passando o valor do i-ésimo elemento dos vetores A e B para a Fórmula 4.2 em cada iteração, ao final, teremos um valor y onde $0 \leq y \leq 2$. A escolhe deste limite 2 foi tomada para evitar que o período entre uma coleta e outra aumente rapidamente, possibilitando assim que várias avaliações do ambiente sejam feitas antes de chegar em longos períodos entre cada coleta de dados, uma forma de evitar configurações radicais.

$$coeficienteFrequencia(A, B, h) = \prod_{i=1}^{h} dif(A^{(i)}, B^{(i)}, h) \qquad (4.1)$$

A Fórmula 4.2 receberá 3 valores, sendo eles:

- a e b - médias de uma métrica obtidas em diferentes períodos;
- h - número de métricas monitoradas pelo serviço.

$$dif(a, b, h) = minimo(a, b)/maximo(a, b) * \sqrt[h]{2} \qquad (4.2)$$

A Fórmula 4.3 receberá 2 valores, sendo eles:

- x - coeficiente de frequência de captura de dados derivado com a Fórmula 4.1;
- y - intervalo de captura de dados inicialmente definida.

A Fórmula 4.3 calcula de fato a frequência de captura de dados do serviço usando a variável x, que é obtido na Fórmula 4.1, e a variável y que é configurada pelo administrador da plataforma. Esta fórmula resulta num valor que pode ser até cem vezes maior ou até duas vezes menor que o valor de y, ou seja, com essa fórmula, o intervalo δ entre cada captura de dados de um serviço pode variar de $y/2 \leq \delta \leq 100?y$. Vale lembrar que esta fórmula usa como valor de entrada o coeficiente calculado na Fórmula 4.1, que como mencionado, pode escalar em no máximo duas vezes e com um mínimo não definido. Com isso, esta fórmula tende a ter um comportamento onde o limite inferior não varia tanto do valor inicial porém pode ser rapidamente alcançado, enquanto o limite superior pode chegar a valores cem vezes maiores que o valor inicial porém numa velocidade menor.

$$intervalo(x, y) = (((x * 99)/(x + 256)) + ((x + 1)/(x + 2))) * y \qquad (4.3)$$

Além do agente de configuração da frequência de coleta de dados dos serviços de captura, o componente de configuração possui também um agente de distribuição de servidores para serviços de captura de dados. Este agente é responsável por verificar a carga de cada serviço e com base nessa carga, decidir quais servidores serão monitorados por cada serviço. Esta carga será uma relação entre o tempo que o servição leva para obter as métricas de um servidor e suas máquinas virtuais e o intervalo de tempo em que este serviço deve disponibilizar as métricas obtidas. O procedimento que o agente de distribuição de servidores para serviços de captura de dados irá usar está descrito no Algoritmo 2.

O Algoritmo 2 tem como objetivo distribuir melhor as cargas do monitoramento dos servidores entre os serviços de captura de dados disponíveis e, quando for necessário este agente irá instanciar mais serviços de captura de dados. Para realizar tal procedimento o Algoritmo 2 tem como entrada uma lista com os serviços de captura de dados disponíveis no ambiente e outra lista com os servidores sem monitoramento [Linha: 1], esta lista de servidores sem monitoramento irá inicializar vazia e será preenchida a medida que os servidores são removidos da lista de servidor a ser monitorado dos serviços de captura de dados. Exceto na inicialização da plataforma, neste momento todos os servidores estarão sem monitoramento, logo esta lista estará cheia e o tempo de captura dos serviços serão todos 0, a lista de serviços conhecidos será preenchida no momento que os serviços de captura realizam seu cadastro na plataforma. As métricas que deverão ser monitoradas de cada servidor na inicialização do sistema, serão configuradas pelo administrador na base de dados da aplicação. Portando o administrador deverá ter ciência das métricas suportadas pelos serviços de captura de dado disponíveis para evitar que servidores não sejam monitorados.

O algoritmo percorre cada serviço de captura de dados recolhendo informações sobre o tempo médio que o serviço está levando para realizar a captura de suas métricas e a frequência de captura configurada no mesmo, com esses 2 valores, o algoritmo pode identificar serviços sobrecarregados ou com pouca carga [Linhas 3 a 17]; no caso de pouca carga, será adicionado um novo servidor sem monitoramento (caso exista) para o serviço de captura de dados passar a monitorar [Linhas 9 a 16].

Caso o serviço de captura de dados esteja sobrecarregado, será então removido 1 servidor de sua lista de servidores a serem monitorados [Linhas 6 a 8]. Um serviço é definido como sobrecarregado quando o tempo médio para realizar a captura das métricas é superior

a frequência com que o serviço deve capturar os dados, o serviço é definido com pouca carga caso este tempo médio de captura seja inferior a metade da frequência com que o serviço irá capturar os dados.

Definidos os serviços de captura de dados disponíveis, o próximo passo do algoritmo será a migrar os servidores de serviços disponíveis menos saturados para serviços disponíveis equivalentes mais saturados [Linhas 19 a 27]. Esta ação irá a permitir esvaziar alguns serviços e aproveitar ao máximo a capacidade de processamento de outros serviços, podendo assim remover os serviços vazios da plataforma como é feito adiante.

Após realizar o balanceamento dos serviços de monitoramento, se ainda existirem servidores não monitorados e não existirem serviços de captura vagos, será então instanciado um novo serviço de captura de dados para suprir os servidores sem monitoramento. O procedimento acaba quando não existirem servidores sem monitoramento, monitorar um servidor inclui monitorar todas as máquinas virtuais existentes naquele servidor [Linhas 28 a 34].

Quando não existirem mais servidores sem monitoramento, será verificado se existe algum serviço de captura de dados sem monitorar nenhum servidor, caso exista, este será removido da plataforma [Linhas 35 a 40].

Descrição de cada variável e métodos presentes no Algoritmo 2:

- *servicosConhecidos*[Linha: 1] - lista com todos os serviços de captura de dados existentes no ambiente;

- *servidoresNaoMonitorados*[Linha: 1] - lista onde cada elemento é o identificador de um servidor e as métricas desse servidor que devem ser monitoradas;

- *servicosDisponiveis*[Linha: 2] - serviços que podem ser designados para monitorar algum servidor caso sejam compatíveis;

- *servico* - um serviço de captura de dados qualquer;

- **obterTiposMetricasMonitoradas**()[Linha: 4] - método que retorna uma lista com todas as métricas que o serviço de captura de dados monitora;

- *tempoGasto*[Linha: 5] - tempo em que um serviço de captura de dados leva para obter o valor das métricas de todos os seus servidores e máquinas virtuais;

- *frequencia*[Linha: 6] - a frequência com que o serviço de captura de dados realiza a coleta dos dados dos servidores e máquinas virtuais pelos quais ele é responsável;

- **adicionar**()[Linha: 7] - função que adiciona um servidor e as métricas que devem ser monitoradas na lista de servidores não monitorados;

- **removeServidor**()[Linha: 7] - método que remove e retorna um servidor da lista de servidores a monitorar de um serviço de captura de dados;

- **procurarServidor**()[Linha: 10] - procura por um servidor na lista de servidores não monitorados que possa ser monitorado pelo serviço passado por parâmetro, se encontrado, o servidor será removido da lista e retornado, senão será retornando vazio. Um servidor pode ser monitorado por um serviço se este serviço suporta algumas das métricas do antigo serviço que monitorava o servidor;

- **adicionarServidor**()[Linha: 14] - adiciona um servidor passado por parâmetro na lista de servidores que serão monitorados pelo serviço de captura de dados;

- **obterServicoEquivalente**()[Linha: 21] - método que retorna um serviço que monitore métricas equivalentes a do serviço passado por parâmetro e que esteja dentro da lista de serviços também passada por parâmetro;

- **transfereServidor**()[Linha: 23] - método que irá remover um servidor a ser monitorado do serviço menos carregado (com menor diferença entre o período em que os dados devem ser obtidos e o tempo levado para obter os dados) e adicionar no mais carregado;

- *alterados*[Linha: 24]- lista contendo serviços disponíveis que já sofreram alterações na sua lista de servidores;

- **obterServidorEmetricas**()[Linha: 29] - método que retorna um objeto servidor qualquer e as métricas que devem ser monitoradas desse servidor;

- **procurarServicos**()[Linha: 32] - função que irá procurar um conjunto de serviços cuja união das métricas monitoradas por esses serviços seja igual às métricas passadas por parâmetro. Se não

existirem serviços suficientes para suprirem todas as métricas, novos serviços serão instanciados, isto garante que todas as métricas serão monitoradas. Os serviços retornados serão removidos da lista de serviços disponíveis mesmo que estes possam suportar mais um servidor para monitorar (o algoritmo considera que o serviço disponível tem capacidade para apenas mais um servidor), se por ventura o serviço de captura ainda tiver capacidade para monitorar mais um servidor, este pode ser designado para monitorar mais um servidor na próxima vez que o algoritmo for executado;

- **adicionarTodos**()[Linha: 33] - método que adiciona um servidor na lista de servidores a monitorar de um conjunto de serviços de captura de dados, além de remover estes serviços da lista de serviços disponíveis e o servidor da lista de servidores sem monitoramento;

- **obterServidores**()[Linha: 36] - método que retorna uma lista com todos os servidores que um serviço de captura de dados monitora;

- **destruir**() [Linha: 38]- método que destrói a instância de um serviço de captura de dados (este método é chamado quando o serviço não possui nenhum servidor para monitorar).

Algorithm 2 alocarServidoresParaServiços

```
1: Entrada: ServicosConhecidos sc, ServidoresNaoMonitorados h
2: ServicosDisponiveis sd
3: for all servico s em sc do
4:    metricas←s.obterTiposMetricasMonitoradas()
5:    tempoGasto←s.tempoGasto()
6:    if tempoGasto>frequencia then
7:       h.adicionar(s.removerServidor(), metricas)
8:    else
9:       if tempoGasto*2 <frequencia then
10:         servidor←procurarServidor(h,metricas)
11:         if Servidor=nulo then
12:            sd.add(s)
13:         else
14:            s.adicionarServidor(servidor)
15:         end if
16:      end if
17:   end if
18: end for
19: for all servicoDisponível s em sd do
20:   if alterados.nãoContem(s) then
21:      servicoEquivalente←obterServicoEquivalente(s,sd)
22:      if servicoEquivalente ≠ nulo then
23:         transfereServidor(s,servicoEquivalente)
24:         alterados.adiciona(servicoEquivalente)
25:      end if
26:   end if
27: end for
28: while h.NaoVazia() do
29:   servidorEmetrica←h.obterServidorEmetricas()
30:   metricas←ServidorEmetrica.metricas
31:   servidor←ServidorEmetrica.servidor
32:   servicos←procurarServicos(sd,metricas)
33:   adicionarTodos(servicos, servidor, h, sd)
34: end while
35: for all servico s em sc do
36:   servidores ← s.obterServidores()
37:   if servidores.Vazio() then
38:      destruir(s)
39:   end if
40: end for
```

O Algoritmo 2 não realiza um balanceamento em cada serviço analisando quantos servidores podem ser adicionados ou removidos, busca-se apenas a adição ou remoção de um servidor em cada serviço e conforme sucessivas execuções do algoritmo, os serviços tenderão a ficar distribuídos.

4.3 FLUXOS DE EXECUÇÃO

A plataforma é constituída de 3 principais fluxos de interação entre os seus componentes. (I) O fluxo de captura de dados que além de obter os dados de cada máquina virtual, persiste os mesmos num repositório de dados; (II) fluxo de configuração de frequência de captura dos dados que irá melhorando a frequência com que cada serviço do componente de monitoramento obtém os valores das métricas dos servidores e suas máquinas virtuais; (III) o fluxo de distribuição de servidores para serviços de captura de dados que irá decidir quais servidores serão monitorados por cada instância dos serviços de captura.

No fluxo de captura de dados ilustrado na Figura 10, cada serviço do componente de monitoramento irá percorrer a sua lista de servidores e para cada servidor, irá capturar o uso de recursos de cada máquina virtual através da API do virtualizador ao mesmo tempo em que será mensurado o tempo utilizado para realizar tal coleta. Após realizar e temporizar as coletas, o componente de monitoramento então irá persistir as informações obtidas no componente de armazenamento. Cada serviço do componente de monitoramento irá atuar em uma fatia de tempo diferente, pois cada um possui sua própria frequência de obtenção de dados.

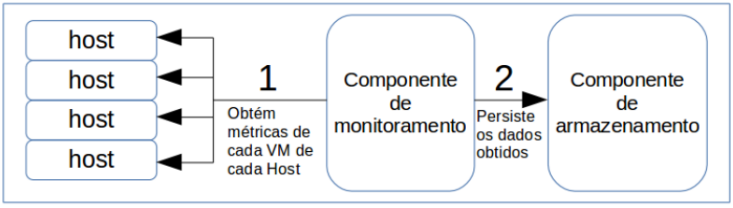

Figura 10 – Fluxo de captura dos dados

No fluxo de configuração da frequência de captura dos dados ilustrado na Figura 11, o componente de configuração irá coletar os dados do ambiente relativos a 2 períodos diferentes, e com esses dados

ele irá redefinir a frequência com que cada serviço do componente de monitoramento deve coletar os dados dos servidores e máquinas virtuais, aumentando ou diminuindo a frequência de coleta dependendo da variação dos dados obtidos pelo serviço nesses 2 períodos distintos, conforme apresentado no Algoritmo 1.

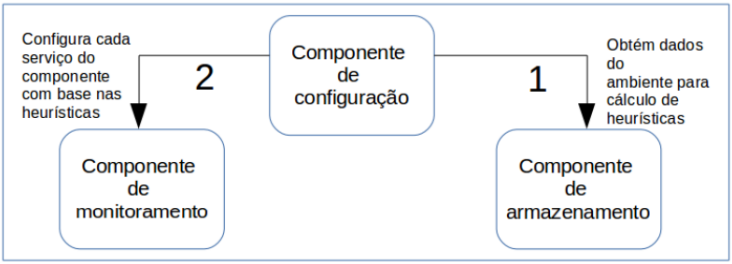

Figura 11 – Fluxo de configuração da frequência de captura dos serviços do componente de monitoramento

No fluxo de distribuição de servidores para serviços de captura de dados apresentado na Figura 12, o componente de configuração irá percorrer todos os serviços do componente de monitoramento e irá consultar o tempo que o serviço leva para concluir sua tarefa de monitoramento e a frequência que a mesma deve ser realizada. Com essas duas métricas, o componente de configuração define se este serviço está sobrecarregado, normal ou pouco carregado. Se o serviço estiver pouco carregado, o componente de configuração irá tentar alocar mais um servidor para o serviço monitorar. Se o serviço estiver sobrecarregado, o componente de configuração irá tentar migrar um servidor deste serviço para outro que realize o mesmo trabalho, caso não seja possível, o componente de configuração irá instanciar um novo serviço para realizar o monitoramento deste servidor a ser migrado. Esse procedimento completo e em detalhes é descrito no Algoritmo 2.

A proposta apresentada utiliza de uma arquitetura modularizada que permite que as funcionalidades da plataforma sejam totalmente independentes, tornando-a extensível e menos suscetível a falhas. A escolha deste tipo de arquitetura também permite a adição de novas funcionalidades para a plataforma de monitoramento em tempo de execução.

Além disso, o uso de agentes para tratar das configurações da plataforma permite maior independência em relação a necessidade de intervenções de administradores para a gerência do sistema. Com isso

a responsabilidade de gerenciar e configurar a plataforma fica a cargo dos agentes e não de administradores humanos, possibilitando com que o sistema se adapte ao ambiente em que ele se encontra.

Outro ponto interessante é o fato da plataforma usar o protocolo HTTP e a representação de objeto JSON para troca de mensagens, o que permite que a implementação dos serviços seja independente de linguagem de programação, ou seja, pode-se ter serviços em diferentes linguagens de programação, como Java, C, Perl e outras. Isto torna a plataforma receptível para qualquer tipo de extensão em qualquer linguagem e *hardware*.

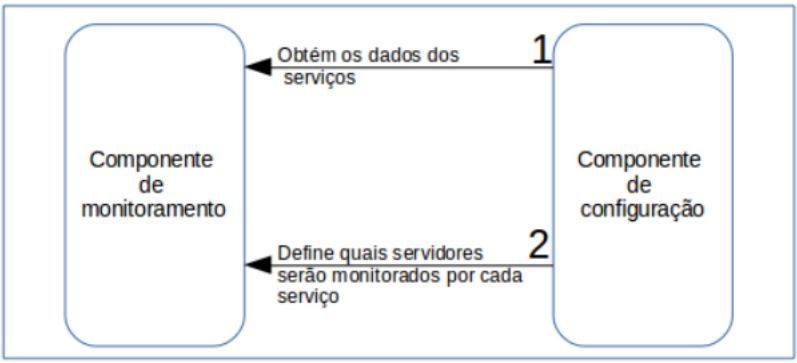

Figura 12 – Fluxo de distribuição de servidores para serviços de captura de dados

5 IMPLEMENTAÇÃO

Este capítulo trata da implementação da plataforma de monitoramento na plataforma de orquestração Apache CloudStack desde as tecnologias usadas até o desenvolvimento da plataforma autônoma de monitoramento e a integração com o orquestrador. Parte do que será descrito neste capítulo já existia de outros trabalhos e foi reaproveitado para integração com o Apache CloudStack.

Como linguagem de programação foi utilizada o Java (ORACLE., 2015a) por ser a mesma linguagem utilizada pelo Apache CloudStack, além de ser bem dominada pelo autor desta proposta, o que poupa tempo com aprendizado de novas linguagens de programação. Para fazer tanto a gerência das instâncias dos novos objetos criados pela plataforma de monitoramento, e a utilização de instâncias de objetos criados pelo Apache CloudStack, foi utilizado o Spring Framework (PIVOTAL., 2016b).

Para realizar a coleta de informações dos servidores e de suas máquinas virtuais, foi utilizada a API do virtualizador Xen (INC., 2015). Após a coleta das informações, foi utilizado o MongoDB (MONGODB., 2016) para realizar a persistência dos dados capturados por ser uma base de dados não relacional bem conhecida e de fácil usabilidade. Por fim, para realizar a criação de *webservices* por onde seria possível fazer o acesso às aplicações, foi utilizado o Spring Boot (PIVOTAL., 2016a) pelo fato de ser uma tecnologia de rápido aprendizado e de desenvolvimento ágil.

A Figura 13 ilustra a hierarquia de projetos que foram usados/-criados para a realização deste trabalho. Como podemos ver, não existem setas ligando o Java e o Spring, pois ambas as tecnologias estão presentes em todos os projetos, logo não há a necessidade de uma ligação em cada projeto com estas tecnologias. Na Figura 13, as setas em vermelho representam relações de dependência de compilação entre projetos, ou seja, o projeto filho herda do projeto pai, somente parâmetros de configuração de compilação e não classes.

As setas pretas por sua vez, indicam dependências de classes ou de funcionalidades, ou seja, o projeto filho depende de uma ou mais classes do projeto pai ou de um ou mais serviços do projeto pai. Serviços neste caso são requisições HTTP que uma aplicação de um projeto faz para outra aplicação de outro projeto.

Outra observação sobre a Figura 13 é a existência do MySQL (ORACLE., 2015b), mesmo ele não tendo sido citado como uma tecno-

logia utilizada na plataforma, isto ocorre pelo fato que ele é utilizado por um projeto que não foi criado juntamente com a plataforma de monitoramento, assim não se teve nenhuma iteração com a tecnologia MySQL. Porém a figura contêm ela apenas para evidenciar sua existência no projeto como um todo.

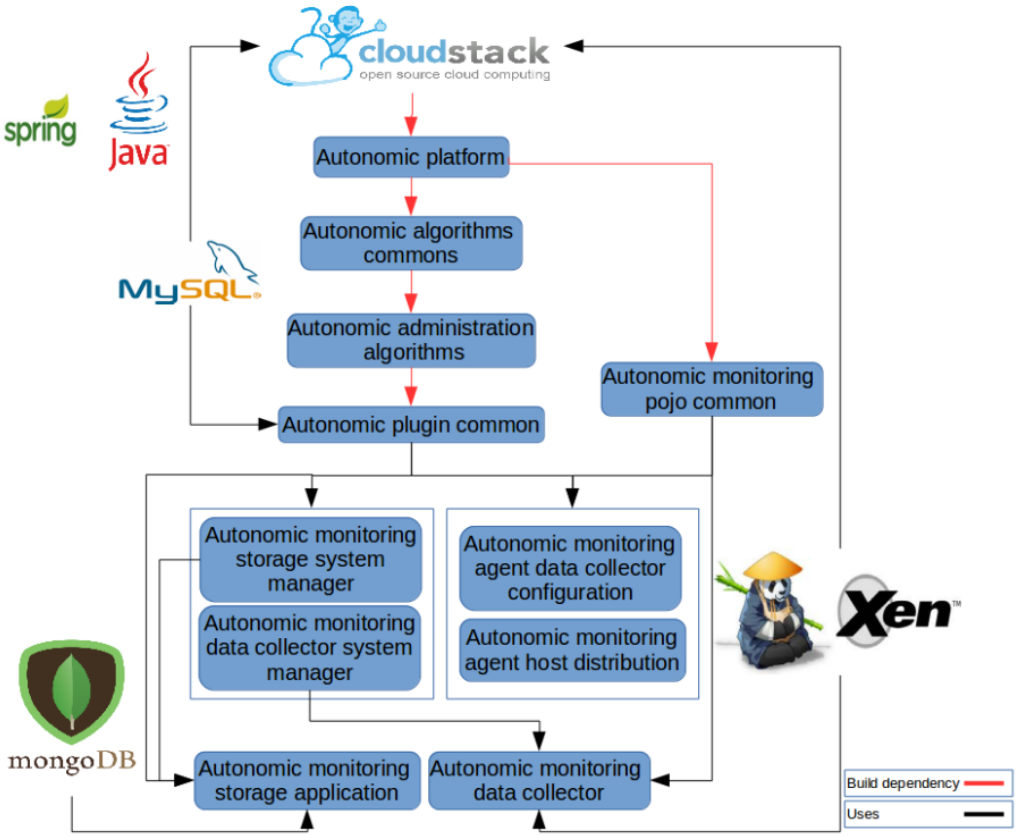

Figura 13 – Hierarquia de projetos da plataforma autônoma de monitoramento

Dentre os projetos já existentes e utilizados temos:

- Autonomic platform, Autonomic algorithms commons e Autonomic administration algorithms - estes projetos foram meramente

utilizados apenas para realizar a compilação da plataforma;

- Autonomic plugin common - foi utilizado deste projeto, instâncias das classes *HostService* - responsável por disponibilizar uma lista com todos os servidores ativos do ambiente; *HttpUtils* - que realiza requisições HTTP POST e GET; *SshUtils* - que realiza comandos em *secure shell* num servidor ou máquina virtual qualquer; *AutonomiccsSystemVmDeploymentService* - essa classe é responsável por realizar a procura de um servidor no ambiente para fazer a hospedagem de uma máquina virtual de sistema e instanciar uma máquina virtual de sistema num servidor escolhido para hospedar os serviços de monitoramento e agentes de configuração.

5.1 PROJETOS CRIADOS PARA A IMPLEMENTAÇÃO DA PLATAFORMA DE MONITORAMENTO

Esta seção descreve cada projeto que compõem a plataforma autônoma de monitoramento.

5.1.1 Autonomic monitoring pojo common

Este projeto serve para padronizar as estruturas dos *Plain Old Java Objects* (POJO) , que são objetos javas simples, sem lógica, com apenas métodos *get* e *set* e o construtor padrão do Java, sua única função é de armazenar e transportar variáveis. Estas estruturas são descritas no modelo entidade relacional apresentado na Figura 8.

5.1.2 Autonomic monitoring storage system manager

Este projeto tem como objetivo criar uma máquina virtual no ambiente, instalar o MongoDB e a aplicação de armazenamento nesta máquina virtual, além de verificar periodicamente se a máquina virtual e a aplicação estão funcionando normalmente.

Para hospedar a máquina virtual criada, primeiramente será preciso de um servidor com capacidade suficiente para suportar a máquina virtual. Deste modo é utilizado o método *searchForRandomHostInCloudToDeployAutonomiccsSystemVm* da *classeAutonomiccsSystemVmDeploymentService*, que irá a procurar por um servidor qualquer no ambiente que suporte uma máquina virtual de sistema como ilustrado

na Figura 14 número 1. Tendo o servidor para hospedar a máquina virtual, o próximo passo é subir a máquina virtual no servidor usando o método *deploySystemVmWithJava* também da classe *AutonomiccsSystemVmDeploymentService*, que irá alocar e iniciar a máquina virtual no servidor informado por parâmetro e instalar o Java nesta máquina virtual, depois disso irá retornar a instância que representa a máquina virtual recentemente criada como ilustrado na Figura 14 número 2.

Com a máquina virtual executando no servidor, é instalada a aplicação de armazenamento. Esta instalação ocorre através de comandos *secure shell* usando a classe *SshUtils*, o endereço de IP da máquina virtual de sistema e uma chave de criptografia que foi configurada no *template* da máquina virtual de sistema. Após ter instalado a aplicação de armazenamento na máquina virtual e ter configurado ela como serviço no sistema operacional da VM, esta então é reiniciada. Esta VM jamais poderá ser destruída, pois isso acarretaria na perda de todo o histórico de monitoramento do ambiente.

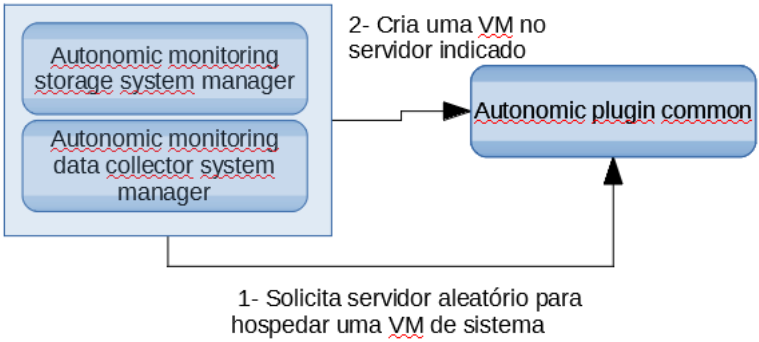

Figura 14 – Fluxo de criação de *System VMs*

Para verificar se a máquina virtual e a aplicação ainda estão em operação como mostrado na Figura 15 número 1, é analisado o estado da máquina virtual no Apache CloudStack, se o estado estiver como Running, então são enviados pings para as portas 8080(aplicação), 27017(MongoDB) e 22 (ssh), se algum desses pings não retornar, então será feita a reinicialização da máquina virtual como ilustrado na Figura 15 número 2. Se o estado estiver diferente de Running, então será a feita a tentativa de inicialização da máquina virtual. Em caso de nenhuma das tentativas funcionarem, então será destruída a máquina virtual e instanciada uma nova (Figura 15, número 3). Para evitar transtornos

de perda de dados, deverá ser criada uma rotina de backup da base de dados e outra rotina de restauração na nova máquina virtual criada.

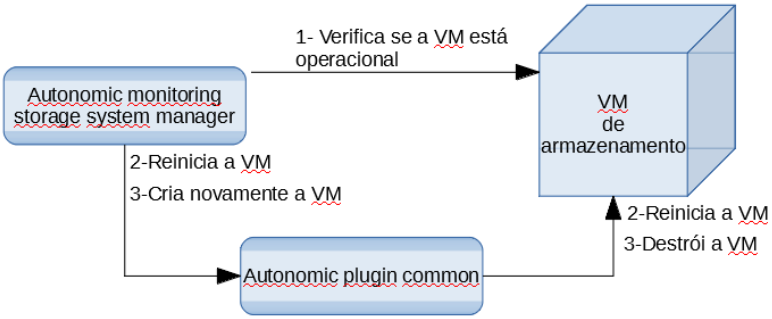

Figura 15 – Fluxo de verificação da VM de armazenamento

5.1.3 Autonomic monitoring data collector system manager

Este projeto é similar ao anterior, o que difere são as aplicações que serão instaladas na máquina virtual de sistema, em vez de instalar a aplicação de armazenamento, será instalada a aplicação de coleta de dados, além disto, este projeto irá permitir que existam mais de uma máquina virtual de coleta de dados no ambiente como ilustrado na Figura 16. Para realizar o monitoramento do funcionamento das máquinas virtuais, será feito um processo similar ao do projeto anterior, diferindo que nestas VMs, a porta 27017 não será monitorada, e caso a máquina virtual não responda ou não possa ser reinicializada, a mesma poderá ser então destruída e dar espaço para uma outra máquina virtual.

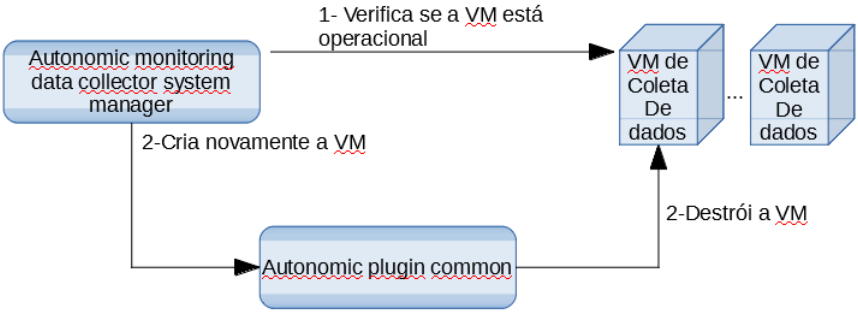

Figura 16 – Fluxo de verificação das VMs de coleta de dados

5.1.4 Autonomic monitoring agent data collector configuration

Este projeto é o responsável por realizar as ações do agente de configuração especificado no capítulo de proposta na seção 4.2.3, além de ser onde é implementado o Algoritmo 1. Este projeto contem duas classes principais *DataCollectorConfigurationAgent* e *DataCollectorConfigurationAgentService*

Na classe DataCollectorConfigurationAgentService é onde foram implementados todos os métodos usados pelo agente que foram especificados no capítulo de proposta seção 4.2.3. Para popular sua lista de serviços de captura de dados esta classe irá fazer requisições *HTTP GET* periódicas para a aplicação de armazenamento da plataforma como ilustrado na Figura 17 número 2.

A classe *DataCollectorConfigurationAgent* irá a usar os métodos implementados pela classe *DataCollectorConfigurationAgentService* para criar um método que implemente o Algoritmo 1. Esta classe que fará o papel do agente de configuração de serviços de coleta de dados proposto. É usado o *Spring Schedule* para fazer a execução periódica do método implementado por esta classe, assim emulando o ciclo de vida de um agente que atua constantemente no ambiente.

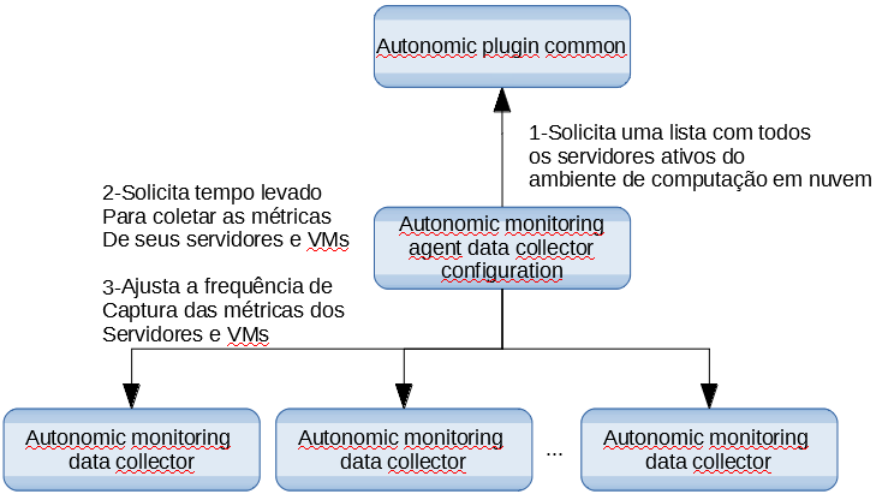

Figura 17 – Fluxo de ajuste da frequência de captura

5.1.5 Autonomic monitoring agent host distribution

Este projeto trata da implementação do agente de distribuição de servidores para serviços de captura de dados proposto. Assim como no projeto anterior, foram criadas duas classes para a implementação deste agente, a classe *HostDistributionAgent* e a classe *HostDistributionAgentService*.

O *HostDistributionAgentService* trata de implementar todos os métodos especificados no Algoritmo 2. Para fazer a detecção de novos servidores adicionados no ambiente de computação em nuvem, esta classe usa uma instância da classe *HostService* do projeto *Autonomic plugin common* que provê um serviço de obtenção de uma lista com todos os servidores ativos existentes no ambiente de computação em nuvem (Figura 18, número 1).

A classe *HostDistributionAgent* usa uma instância da classe *HostDistributionAgentService* para realizar a implementação do Algoritmo 2, além de usar o *Spring Schedule* para fazer a execução periódica do Algoritmo 2, e assim emular o ciclo de vida de um agente.

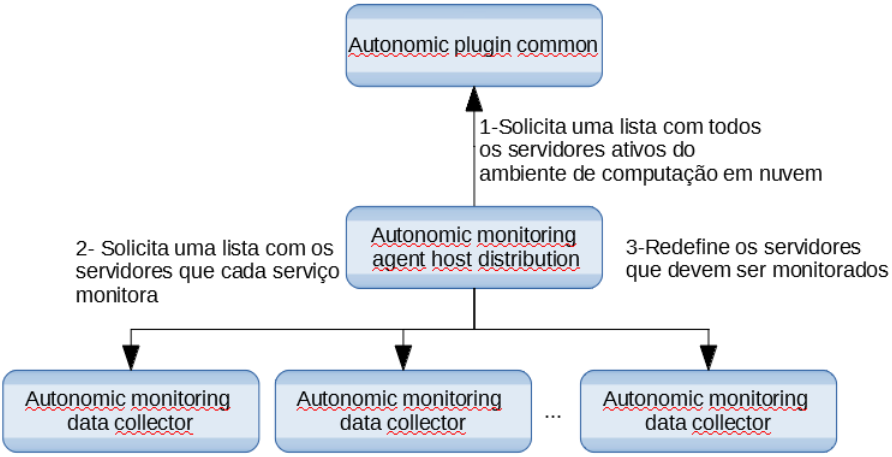

Figura 18 – Fluxo de ajuste da alocação de servidores para serviços de captura

5.1.6 Autonomic monitoring data collector application

Este projeto é responsável pela coleta dos dados do ambiente, para isso ele irá usar a XAPI do Xen, esta API permite a coleta de inúmeros dados das máquinas virtuais e do servidor como mostrado na Figura 19 número 1. Para ter acesso à API, é preciso fornecer um usuário com permissão de administrador e sua senha no servidor. A API retorna um documento XML como resposta às requisições.

Neste projeto existe a classe *DataCollectorEndPoints* que é responsável por tratar os *EndPoints* do servidor *REST* do serviço de captura de dados para o agente de configuração poder requisitar e atualizar as variáveis deste serviço. Este projeto também possui a classe *CaptureService* que irá fazer uso da API do Xen para consultar os dados de cada servidor e máquina virtual pelo qual este serviço é responsável.

A classe *CaptureService* também implementa a interface do Spring *InitializingBean*, esta interface provê o método afterPropertiesSet, este método é executado logo após a criação do objeto que é gerenciado pelo *Spring*. É neste método que o serviço irá se cadastrar na base de dados da plataforma e poderá então ser configurado corretamente pelos agentes da plataforma.

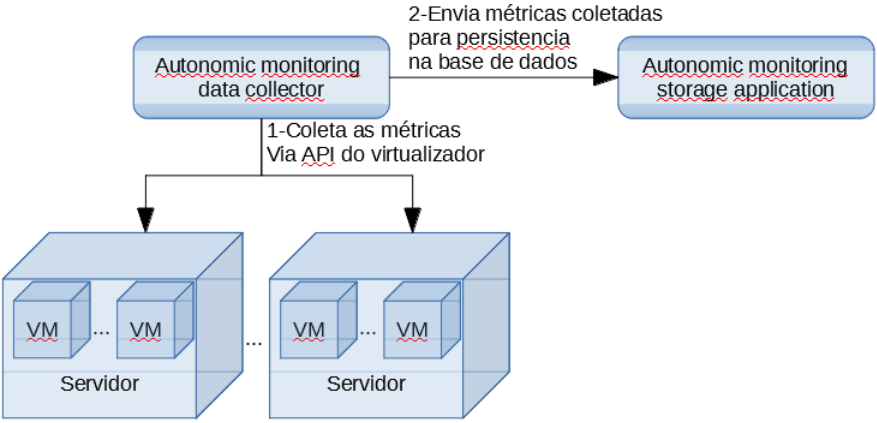

Figura 19 – Fluxo de coleta de dados

5.1.7 Autonomic monitoring storage application

Este projeto é responsável pela criação do serviço de armazenamento da plataforma, assim como por instalar corretamente o MongoDB na VM onde a aplicação for instalada.

Este projeto possui a classe *StorageApplicationEndPoint* que irá tratar as requisições HTTP GET e POST feitas para esta aplicação. A classe *StorageService* trata da realização de consultas e persistências na base de dados através da classe *StorageDao*. Além disso, esta classe implementa o *InitializingBean* e realiza a instalação do MongoDB no método *afterPropertiesSet*, caso necessário.

Para realizar a instala¸ao do MongoDB é usada uma instância da classe *ShellCommandUtils* para a execução de um *ShellScript* que foi inserido na máquina virtual na hora da instalação da aplicação na mesma pelo *Autonomic monitoring storage system manager*.

A classe *StorageDAO* é quem irá fazer a comunicação direta com a base de dados usando para isso a API Java do MongoDB.

6 CONCLUSÃO

Este trabalho de conclusão de curso resultou na criação de um modelo de plataforma de monitoramento de ambientes de computação em nuvem autônoma que pode ser implementada em qualquer ferramenta de orquestração. Contudo, como validação foi criado um plugin, que implementa a proposta e a introduz na ferramenta de orquestração Apache CloudStack.

Foi desenvolvido um plugin para o Apache CloudStack que permite o monitoramento autônomo e criação de um histórico de métricas relativo a alocação e uso de recursos em ambientes orquestrados pelo Apache CloudStack. Para isto foi incluído no fluxo de inicialização do Apache CloudStack, funções que irão criar e instanciar os serviços e agentes necessários para o funcionamento da plataforma de monitoramento autônoma. A integração da plataforma de monitoramento com o Apache CloudStack se deu através do uso de projetos já existentes que continham inúmeras abstrações para uso de funções do Apache CloudStack. Outro ponto sobre a implementação foi o uso da API do Xen para a coleta dos dados das máquinas virtuais e servidores.

O uso de agentes para a configuração autônoma do ambiente resulta num melhor uso dos recursos computacionais para realizar o monitoramento do ambiente sem prejuízos na acurácia dos resultados. Entretanto, devido ao prazo para conclusão do presente trabalho, não foi possível realizar experimentos e mensurar a eficiência, acurácia e outros aspectos da ferramenta proposta.

Este trabalho possui as seguintes contribuições:

- Criação de um modelo para uma plataforma de monitoramento que pode ser replicado em outras ferramentas de orquestração;
- Extensão de uma ferramenta de código livre para aplicação e experimentação da proposta;
- Criação de base de código que pode ser usado para futuras pesquisas na área de monitoramento de ambientes de computação em nuvem.

Em uma breve comparação deste trabalho com outras ferramentas de monitoramento de ambientes de computação em nuvem, tem-se como principal ponto a autonomia da ferramenta desta proposta em relação a das demais soluções existentes. Enquanto as soluções atuais necessitam que os administradores do ambiente realizem ajustes

em seus parâmetros para melhor adequar a ferramenta ao ambiente, esta proposta independe de administradores para alcançar tal façanha, atribuindo esta responsabilidade aos agentes modelados.

Como trabalho futuros têm-se as seguintes atividades:

- Melhorar o código fonte desenvolvido com documentação e test cases;
- Disponibilizar a implementação da plataforma a comunidade;
- Realizar testes comparando esta solução com as já existentes;
- Criar um artigo científico com a proposta desta plataforma.

REFERÊNCIAS

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

ACETO, G. et al. Cloud monitoring: Definitions, issues and future directions. **CLOUDNET**, v. 12, p. 63–67, 2012.

ANDREOZZI, S. et al. Gridice: a monitoring service for grid systems. **Future Generation Computer Systems**, Elsevier, v. 21, n. 4, p. 559–571, 2005.

APACHE. **Apache**. 2015. Acesso em 14 de setembro de 2015. Disponível em: <http://apache.org/>.

APACHE, f. **CloudStack**. 2015. Acesso em 14 de setembro de 2015. Disponível em: <http://cloudstack.apache.org/>.

ARMBRUST, M. et al. A view of cloud computing. **Communications of the ACM**, ACM, v. 53, n. 4, p. 50–58, 2010.

BEN, A. et al. **System x Virtualization Strategies**. [s.n.], 2010. Disponível em: <https://lenovopress.com/redp4480>.

BRäSCHER, G. B. **PROPOSTA DE UM FRAMEWORK PARA CONSOLIDAÇÃO DE RECURSOS EM AMBIENTES DE COMPUTAÇÃO EM NUVEM**. 2015.

CANONICAL. **Canonical**. 2015. Acesso em 24 de novembro de 2015. Disponível em: <http://www.canonical.com/about>.

CARON, E. et al. Auto-scaling, load balancing and monitoring in commercial and open-source clouds. 2012.

CLAYMAN, S.; GALIS, A.; MAMATAS, L. Monitoring virtual networks with lattice. In: IEEE. **Network Operations and Management Symposium Workshops (NOMS Wksps), 2010 IEEE/IFIP**. [S.l.], 2010. p. 239–246.

CLOUDSTACKZENPACK. **Zenoss**. 2015. Disponível em: <https://github.com/zenoss/ZenPacks.zenoss.CloudStack>.

CONTAINERS., L. **LXC**. 2015. Acesso em 24 de novembro de 2015. Disponível em: <https://linuxcontainers.org/lxc/introduction/>.

CORPORATION, I. **IA 32**. 2015. Acesso em 15 de outubro de 2015. Disponível em: <http://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-manual-325462.pdf>.

CORPORATION, I. **x8664**. 2015. Acesso em 15 de outubro de 2015. Disponível em: <http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html>.

DOSHI, G. **IA 64**. 2015. Acesso em 15 de outubro de 2015. Disponível em: <http://www.csee.umbc.edu/portal/help/architecture/idfisa.pdf>.

GLOBUS. **Globus Toolkit**. 2015. Acesso em 17 de dezembro de 2015. Disponível em: <http://toolkit.globus.org/toolkit/mds/>.

HALL, P. Opportunities for csps in enterprise-grade public cloud computing. **OVUM, May**, 2012.

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

HP. **Eucalyptus**. 2015. Acesso em 14 de setembro de 2015. Disponível em: <http://www8.hp.com/us/en/cloud/helion-eucalyptus-overview.html>.

HP. **HP**. 2015. Acesso em 29 de setembro de 2015. Disponível em: <http://www.hp.com/>.

HYPERIC. **Hyperic**. 2015. Acesso em 18 de dezembro de 2015. Disponível em: <https://support.hyperic.com/display/SIGAR/Home>.

IBM. **Power PC**. 2015. Acesso em 15 de outubro de 2015. Disponível em: <http://www.ibm.com/developerworks/systems/library/es-archguide-v2.html>.

ICINGA. **Icinga**. 2015. Acesso em 30 de novembro de 2015. Disponível em: <https://www.icinga.org/>.

ICINGAOPENSTACKGIT. **OpenStack-Icinga**. 2015. Acesso em 30 de outubro de 2015. Disponível em: <https://github.com/hpcloud/icinga>.

INC., C. S. **Xen Project Software Overview**. 2015. Acesso em 10 de novembro de 2015. Disponível em: <http://wiki.xenproject.org/wiki/Xen_Overview>.

INC., O. **ORACLE**. 2015. Acesso em 10 de novembro de 2015. Disponível em: <http://www.oracle.com/>.

INTEL. **IPMI**. 2015. Acesso em 29 de novembro de 2015. Disponível em: <http://www.intel.com/content/www/us/en/servers/ipmi/ipmi-home.html>.

KATSAROS, G. et al. A self-adaptive hierarchical monitoring mechanism for clouds. **Journal of Systems and Software**, Elsevier, v. 85, n. 5, p. 1029–1041, 2012.

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

KEPHART, J. O.; CHESS, D. M. The vision of autonomic computing. **Computer**, IEEE, v. 36, n. 1, p. 41–50, 2003.

LAPRIE, J.-C. From dependability to resilience. In: CITESEER. **38th IEEE/IFIP Int. Conf. On Dependable Systems and Networks**. [S.l.], 2008. p. G8–G9.

LI, Y.; MANOHARAN, S. A performance comparison of sql and nosql databases. In: IEEE. **Communications, Computers and Signal Processing (PACRIM), 2013 IEEE Pacific Rim Conference on**. [S.l.], 2013. p. 15–19.

LINUX-KVM. **KVM**. 2015. Acesso em 14 de setembro de 2015. Disponível em: <http://www.linux-kvm.org/page/Main_Page>.

MAZEGEN. **x86**. 2007. Acesso em 15 de outubro de 2015. Disponível em: <http://x86asm.net/articles/x86-64-tour-of-intel-manuals/>.

MELL, P.; GRANCE, T. The nist definition of cloud computing. Computer Security Division, Information Technology Laboratory, National Institute of Standards and Technology Gaithersburg, 2011.

MENASCÉ, D. A. Virtualization: Concepts, applications, and performance modeling. In: **Int. CMG Conference**. [S.l.: s.n.], 2005. p. 407–414.

MICROSOFT. **NFS**. 2013. Acesso em 18 de novembro de 2015. Disponível em: <https://technet.microsoft.com/en-us/library/jj592688.aspx>.

MICROSOFT. **Hyper-V**. 2015. Acesso em 29 de setembro de 2015. Disponível em: <http://hyperv.veeam.com/what-is-hyper-v-technology/>.

MICROSOFT. **iSCSI**. 2015. Acesso em 18 de novembro de 2015. Disponível em: <http://windows.microsoft.com/pt-br/windows-vista/what-is-internet-small-computer-system-interface-iscsi>.

MONGODB. **MongoDB**. 2016. Acesso em 6 de junho de 2016. Disponível em: <https://docs.mongodb.com>.

NAGIOS. **Nagios**. 2015. Acesso em 14 de setembro de 2015. Disponível em: <https://assets.nagios.com/downloads/nagioscore/docs/nagioscore-3-en.pdf>.

NAGIOS. **Nagios-CloudStack**. 2015. Acesso em 13 de novembro de 2015. Disponível em: <https://exchange.nagios.org/directory/Plugins/Cloud/nagios-2Dcloudstack/details>.

NAGIOS. **Nagios-OpenStack**. 2015. Acesso em 13 de novembro de 2015. Disponível em: <https://exchange.nagios.org/directory/Plugins/Cloud/openstack-infrastructure-monitoring/details>.

NET-SNMP. **SNMP**. 2015. Acesso em 30 de outubro de 2015. Disponível em: <http://www.net-snmp.org/>.

OPENSTACK. **OpenStack**. 2015. Acesso em 14 de setembro de 2015. Disponível em: <http://docs.openstack.org/index.html>.

OPENSTACKZENPACK. **Zenoss**. 2015. Acesso em 30 de outubro de 2015. Disponível em: <http://wiki.zenoss.org/ZenPack:OpenStack_(Provider_View)>.

ORACLE. **Java**. 2015. Acesso em 6 de junho de 2016. Disponível em: <https://www.java.com/pt_BR/about/whatis_java.jsp>.

ORACLE. **MySQL**. 2015. Acesso em 6 de junho de 2016. Disponível em: <http://dev.mysql.com/doc>.

ORACLE. **VirtualBox**. 2015. Acesso em 10 de novembro de 2015. Disponível em: <https://www.virtualbox.org/wiki/Documentation>.

PIVOTAL. **Spring Boot**. 2016. Acesso em 6 de junho de 2016. Disponível em: <http://projects.spring.io/springboot/>.

PIVOTAL. **Spring Framework**. 2016. Acesso em 6 de junho de 2016. Disponível em: <https://spring.io/projects>.

POPEK, G. J.; GOLDBERG, R. P. Formal requirements for virtualizable third generation architectures. **Communications of the ACM**, ACM, v. 17, n. 7, p. 412–421, 1974.

POSTGRESQL. **PostgreSQL**. 2015. Acesso em 17 de dezembro de 2015. Disponível em: <http://www.postgresql.org/>.

PROJECT, N. **Nimbus**. 2015. Acesso em 30 de outubro de 2015. Disponível em: <http://www.nimbusproject.org/>.

QEMU. **QEMU**. 2015. Acesso em 7 de outubro de 2015. Disponível em: <http://wiki.qemu.org/Main_Page>.

REDHAT. **RedHatVirtualization**. 2015. Acesso em 14 de setembro de 2015. Disponível em: <http://www.redhat.com/en/technologies/virtualization/enterprise-virtualization>.

SAHOO, J.; MOHAPATRA, S.; LATH, R. Virtualization: A survey on concepts, taxonomy and associated security issues. In: IEEE. **Computer and Network Technology (ICCNT), 2010 Second International Conference on**. [S.l.], 2010. p. 222–226.

SHAO, J. et al. A runtime model based monitoring approach for cloud. In: IEEE. **Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on**. [S.l.], 2010. p. 313–320.

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

SINGH, G.; SOOD, S.; SHARMA, A. Cm-measurement facets for cloud performance. **International Journal of Computer Applications**, International Journal of Computer Applications, 244 5 th Avenue,# 1526, New York, NY 10001, USA India, v. 23, n. 3, p. 37–42, 2011.

SYSTEMS, C. **Citrix**. 2015. Acesso em 14 de setembro de 2015. Disponível em: <http://www.citrix.com/>.

SYSTEMS, C. **Xen Dom0**. 2015. Acesso em 15 de outubro de 2015. Disponível em: <http://wiki.xenproject.org/wiki/Dom0>.

SYSTEMS, C. **Xen DomU**. 2015. Acesso em 15 de outubro de 2015. Disponível em: <http://wiki.xenproject.org/wiki/DomU>.

SYSTEMS, C. **Xen XAPI**. 2015. Acesso em 15 de outubro de 2015. Disponível em: <http://wiki.xenproject.org/wiki/XAPI>.

SYSTEMS, C. **Xen XL**. 2015. Acesso em 15 de outubro de 2015. Disponível em: <http://wiki.xenproject.org/wiki/XL>.

URGAONKAR, B.; SHENOY, P.; ROSCOE, T. Resource overbooking and application profiling in shared hosting platforms. **ACM SIGOPS Operating Systems Review**, ACM, v. 36, n. SI, p. 239–254, 2002.

VIRATANAPANU, A. et al. On demand fine grain resource monitoring system for server consolidation. In: IEEE. **Kaleidoscope: Beyond the Internet?-Innovations for Future Networks and Services, 2010 ITU-T**. [S.l.], 2010. p. 1–8.

VMWARE. **Understanding Full Virtualization, ParaVirtualization, and Hardware Assist.** 2007. Acesso em 17 de outubro de 2015. Disponível em: <http://www.vmware.com/resources/techresources/1008>.

VMWARE. **ESXI**. 2015. Acesso em 14 de setembro de 2015. Disponível em: <http://www.vmware.com/products/esxi-and-esx/>.

VMWARE. **vCenter**. 2015. Acesso em 10 de novembro de 2015. Disponível em: <http://www.vmware.com/br/products/vcenter-server/>.

VMWARE. **VMware**. 2015. Acesso em 14 de setembro de 2015. Disponível em: <http://www.vmware.com>.

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

WEINGÄRTNER, R.; BRÄSCHER, G. B.; WESTPHALL, C. B. Cloud resource management: A survey on forecasting and profiling models. **Journal of Network and Computer Applications**, Elsevier, v. 47, p. 99–106, 2015.

ZABBIX. **ZABBIX**. 2015. Acesso em 30 de novembro de 2015. Disponível em: <http://www.zabbix.com/product.php>.

ZABBIX-CLOUDSTACK. **Zabbix-Cloudstack**. 2015. Acesso em 30 de outubro de 2015. Disponível em: <https://github.com/ke4qqq/zabbix-cloudstack>.

ZABBIXAGENTADOPTION. **Zabbix agent adoption**. 2015. Acesso em 30 de outubro de 2015. Disponível em: <https://wiki.openstack.org/wiki/Zabbix-agent-adoption>.

ZENOSS. **Zenoss Wiki**. 2015. Acesso em 10 de novembro de 2015. Disponível em: <http://wiki.zenoss.org/Main_Page>.