UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

ARQDEP: ARQUITETURA DE COMPUTAÇÃO EM NUVEM COM DEPENDABILIDADE

GEYCY DYANY OLIVEIRA LIMA

Uberlândia - Minas Gerais
2014

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

GEYCY DYANY OLIVEIRA LIMA

ARQDEP: ARQUITETURA DE COMPUTAÇÃO EM NUVEM COM DEPENDABILIDADE

Dissertação de Mestrado apresentada à Faculdade de Computação da Universidade Federal de Uberlândia, Minas Gerais, como parte dos requisitos exigidos para obtenção do título de Mestre em Ciência da Computação.

Área de concentração: Sistemas Computacionais.

Orientador:
Prof. Dr. Jamil Salem Barbar

Uberlândia, Minas Gerais
2014

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Os abaixo assinados, por meio deste, certificam que leram e recomendam para a Faculdade de Computação a aceitação da dissertação intitulada "**ArqDep: Arquitetura de Computação em Nuvem com Dependabilidade**" por **Geycy Dyany Oliveira Lima** como parte dos requisitos exigidos para a obtenção do título de **Mestre em Ciência da Computação**.

Uberlândia, 03 de Fevereiro de 2014

Orientador: _____
Prof. Dr. Jamil Salem Barbar
Universidade Federal de Uberlândia

Banca Examinadora:

_____
Prof. Dr. Mehran Misaghi
Sociedade Educacional de Santa Catarina

_____
Prof. Dr. Pedro Frosi Rosa
Universidade Federal de Uberlândia

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Data: 03 de Fevereiro de 2014

Autor:	**Geycy Dyany Oliveira Lima**
Título:	**ArqDep: Arquitetura de Computação em Nuvem com Dependabilidade**
Faculdade:	**Faculdade de Computação**
Grau:	**Mestrado**

_____

Autor

Dedicatória

A minha amada e preciosa família.

Agradecimentos

Ao Prof. Dr. Jamil Salem Barbar, meu profundo agradecimento pelos ensinamentos, dedicação, parceria e confiança demonstrada durante o trabalho.

A Capes pelo apoio financeiro dado a este trabalho durante a realização da pesquisa.

A Faculdade de Computação e ao Programa de Pós-Graduação em Ciência da Computação, pela oportunidade de grande crescimento profissional e intelectual.

Ao secretário Erisvaldo pela eficiência com que sempre me atendeu.

Ao Diretor Dr. Ilmério Reis da Silva e a Coordenadora Dr. Denise Guliato que forneceu equipamentos da Pós-Graduação para o desenvolvimento desta pesquisa.

A todos os amigos que encontrei durante a caminhada que de alguma forma colaborou para o êxito desta dissertação. Destaco esta colaboração por meio dos amigos: Alex Eustáquio, Ana Maria, Elder Vicente, Taffarel, Juan, Joicy, Luciane, Newarney, Miguel, Diego, Fabíola, Rafael, Romêrson, Cleiane e Juliete.

Epigrafe

"Tenha coragem de seguir o que seu coração e sua intuição dizem. Eles já sabem o que você realmente deseja. Todo resto é secundário." Steve Jobs

Resumo

A Computação em Nuvem emergiu como uma das mais influentes tecnologias na indústria de TI e está revolucionando o modo pelo qual os recursos de TI são gerenciados e utilizados. Prover ambientes com dependabilidade tornou-se algo importante e difícil para esses ambientes, em virtude da heterogeneidade das infraestruturas. A Computação em Nuvem é tratada como uma tecnologia que requer uma confiança entre o provedor e os consumidores. As definições sobre Computação em Nuvem são discutidas, para melhor entendimento da proposta desta Dissertação de Mestrado. De uma forma clara e sucinta, são definidas as características essenciais da Computação em Nuvem, os modelos de serviços reconhecidos pelo NIST, os modelos de implantação da nuvem, os SLAs e duas arquiteturas de provedores atuantes no mercado são discutidas. Os conceitos de dependabilidade são introduzidos, assim como, as ameaças e os meios de alcançar a dependabilidade. É apresentada uma arquitetura de Computação em Nuvem com dependabilidade, denominada ArqDep. A ArqDep é implementada em uma Infraestrutura como Serviço - IaaS, no modelo de implantação em nuvem privada. Para auxiliar na implementação da arquitetura, utilizou-se uma plataforma de gerenciamento para nuvem, chamada *OpenStack*. Um módulo de gerenciamento de infraestrutura de Computação em Nuvem é proposto: esse módulo realiza o monitoramento e o controle da infraestrutura. Baseando-se no modelo de classificação de falhas é elaborada uma Árvore de Falhas do sistema. Diversas métricas dos recursos computacionais que afetam o desempenho, a eficiência, a disponibilidade e a confiabilidade do sistema são coletadas da infraestrutura e mecanismos responsáveis pela realocação de recursos são utilizados.

Palavras chave: Computação em Nuvem, Dependabilidade, Serviços, Confiabilidade, Disponibilidade.

Abstract

Cloud Computing has emerged as one of the most influential technologies of the IT industry, and is presently revolutionizing the way by which IT resources are used and managed. To test dependability environments becomes something which is both important as well as difficult for these environments, in virtue of the infrastructures heterogeneity. Cloud Computing is treated as a technology that requires a high level of trust between the provider and the client. The definitions concerning Cloud Computing are discussed for a better understanding of the proposal made in this Master's Dissertation. The essential features of Cloud Computing are defined in a clear and brief manner; the service models recognized by NIST, the cloud implementation models, the SLAs along with two provider architectures currently used on the market are also discussed. The concepts behind dependability are introduced, as also are the threats and the means by which dependability can be reached. A dependability based Cloud Computing architecture denominated ArqDep is presented. The ArqDep architecture is implemented into an Infrastructure as a Service - IaaS, in the private Cloud implementation model. To aid in the architecture's implementation a Cloud management platform called OpenStack was used. An infrastructure management model for Cloud Computing is proposed: this model carries out the monitoring and control of the infrastructure. A system failure tree is elaborated, which is based upon the failure classification model. Computer resource metrics that affect availability and confidence in the system are collected from the infrastructure and mechanisms responsible for the relocation of resources are used.

Keywords: Cloud Computing, Dependability, Service, Reliability, Availability.

Sumário

Lista de Figuras **xxi**

Lista de Abreviaturas e Siglas **xxiii**

1 Introdução **25**

2 Computação em Nuvem **31**

2.1 Conceitos Básicos 31

2.1.1 Definições 33

2.1.2 Características Principais da Computação em Nuvem 34

2.2 Modelos de Serviços 36

2.2.1 *Software* como um Serviço (SaaS) 37

2.2.2 Plataforma como um Serviço (PaaS) 38

2.2.3 Infraestrutura como Serviço (IaaS) 38

2.2.4 Gerenciamento de Serviços 39

2.3 Modelos de Implantação de Nuvem 40

2.3.1 Nuvem Privada 41

2.3.2 Nuvem Pública 41

2.3.3 Nuvem Híbrida 42

2.3.4 Nuvem Comunidade 42

2.4 Arquiteturas de Computação em Nuvem 43

2.4.1 A Arquitetura da *Amazon Elastic Compute Cloud (Amazon EC2)* . 43

2.4.2 Arquitetura do *Rackspace* 44

2.5 Acordo de Nível de Serviço (SLA) 48

2.6 Desafios da Computação em Nuvem 49

2.6.1 Disponibilidade do serviço 49

2.6.2 Dados em *lock-in* 50

2.6.3 Confidencialidade e auditabilidade 51

2.6.4 Gargalos na transferência de arquivos 51

2.6.5 Imprevisibilidade no desempenho 51

2.6.6 Armazenamento escalável 51

2.6.7 *Bugs* em larga escala em sistemas distribuídos 52
2.6.8 Escalonamento rápido 52
2.6.9 Lista de reputação 53
2.6.10 Licença de *software* 53
2.7 Conclusão 53

3 Plataformas de gerenciamento e dependabilidade das nuvens 55
3.1 Plataformas de Gerenciamento para Nuvem 55
3.1.1 *Eucalyptus* 56
3.1.2 *OpenNebula* 57
3.1.3 *OpenStack* 59
3.1.4 Versões do *OpenStack* 61
3.1.5 *Dashboard (Horizon)* 61
3.1.6 *Object Store (Swift)* 63
3.1.7 *Image Store (Glance)* 64
3.1.8 *Compute (Nova)* 64
3.1.9 *Identity (Keystone)* 65
3.1.10 *Networking (Neutron)* 67
3.1.11 *Block Storage (Cinder)* 67
3.1.12 *Heat* 68
3.1.13 *Ceilometer* 68
3.1.14 Método de comunicação 71
3.2 Dependabilidade 75
3.2.1 Conceitos Básicos 75
3.3 Atributos da Dependabilidade 76
3.3.1 Confiabilidade 77
3.3.2 Disponibilidade 79
3.4 Ameaças a Dependabilidade 79
3.4.1 Falha 79
3.4.2 Erro 80
3.4.3 Defeito 81
3.5 Meios para Alcançar a Dependabilidade 82
3.5.1 Prevenção a falhas 82
3.5.2 Tolerância a falhas 83
3.5.3 Remoção de falhas 84
3.5.4 Previsão a falhas 84
3.6 Modelos para Avaliação de Dependabilidade 85
3.6.1 Árvore de Falhas 85
3.6.2 Diagramas de Bloco de Confiabilidade 87

3.6.3 Sistema em série . . . 87
3.6.4 Sistema em paralelo . . . 88
3.6.5 Cadeias de Markov . . . 89
3.6.6 Redes de Petri Estocásticas . . . 91
3.7 Conclusão . . . 92

4 A ArqDep **93**
4.1 Visão Geral . . . 96
4.2 Descrição da ArqDep . . . 97
4.2.1 As relações entre os atores da ArqDep . . . 98
4.2.2 As relações entre as camadas do Provedor . . . 100
4.2.3 As camadas do Provedor . . . 101
4.3 Implementação da ArqDep . . . 105
4.4 Infraestrutura como Serviço - IaaS . . . 107
4.4.1 Máquina Controlador . . . 108
4.4.2 Máquinas *Compute* . . . 111
4.4.3 *Hypervisor QEMU* . . . 111
4.4.4 Máquina *Chef-service* . . . 112
4.5 Módulo de gerenciamento . . . 112
4.5.1 Classificação das falhas . . . 115
4.5.2 Realocação de recursos . . . 117
4.6 Avaliação da Dependabilidade . . . 120
4.6.1 Árvore de Falhas . . . 120
4.7 Métricas Coletadas . . . 123
4.7.1 Discussões e resultados . . . 123
4.8 Conclusão . . . 126

5 Considerações Finais **127**

Referências **131**

A Instalação e configuração do MySQL **137**

B Instalação e configuração do RabbitMQ **139**

C Instalação e configuração do *Keystone* **141**

D Instalação e configuração do *Glance* **145**

E Instalação e configuração do *Nova* **149**

F Instalação e configuração do *Swift* **157**

G Instalação e configuração do *Cinder* **163**

H Instalação e configuração do *Ceilometer* **167**

Lista de Figuras

2.1 Linha do tempo da Computação em Nuvem 32
2.2 Pesquisas globais sobre o termo *Cloud Computing* 33
2.3 Modelos de Serviços 37
2.4 Gerenciamento de Serviços 39
2.5 Modelos de implantação de Nuvem 40
2.6 Arquitetura básica do *Amazon Elastic Compute Cloud* 44
2.7 Arquitetura de Nuvem Híbrida de Alta Disponibilidade *Rackspace* 47

3.1 Arquitetura do *Eucalyptus* 57
3.2 Arquitetura do OpenNebula 58
3.3 Arquitetura do *OpenStack* 60
3.4 Tela do *Dashboard* 62
3.5 Tela do *Dashboard* para visualizar os usuários 66
3.6 Tela do *Dashboard* para visualizar os papéis 66
3.7 Tela do *Dashboard* para visualizar os projetos 67
3.8 Arquitetura do *Ceilometer* 69
3.9 Coleta de dados pelo *Collector* e *Agent* 70
3.10 Acesso aos dados armazenados pelo *Ceilometer* 71
3.11 Funcionamento do método *Publisher-Subscriber* 72
3.12 Árvore de Dependabilidade 77
3.13 Modos de falhas de serviços 80
3.14 Classes elementares de defeitos 81
3.15 Cadeia fundamental da dependabilidade 82
3.16 Meios para obter tolerância a falhas 83
3.17 Função de distribuição acumulada das portas lógicas 86
3.18 Sistema em série 88
3.19 Sistema em paralelo 88
3.20 Exemplo Cadeia de Markov com dois estados 90
3.21 Grafo e seus elementos básicos 91
3.22 Exemplo de uma Rede de Petri Estocástica 91

4.1 Arquitetura de referência do NIST 94
4.2 Arquitetura lógica do *OpenStack* 95
4.3 A ArqDep 98
4.4 Relações entre atores da arquitetura 99
4.5 Relações entre as camadas do Provedor 101
4.6 Componentes da camada Orquestração de Serviços 102
4.7 Componentes do Controlador 103
4.8 Funcionalidades dos componentes do *OpenStack* 104
4.9 Componentes da camada Dependabilidade de serviços 105
4.10 Topologia da Nuvem Privada 106
4.11 Módulo de gerenciamento de infraestruturas de Computação em Nuvem . . 113
4.12 Modelo de classificação de falhas 116
4.13 Fluxograma para realocação de recursos 118
4.14 Árvore de falhas do sistema 122
4.15 Monitoramento da utilização da CPU 124
4.16 Monitoramento da memória 125

Lista de Abreviaturas e Siglas

AMI	*Amazon Machine Image*
AMQP	*Advanced Message Queuing Protocol*
API	*Application Programming Interface*
AWS	*Amazon Web Services*
CC	*Cluster Controller*
CDF	*Cumulative Density Function*
CLC	*Cloud Controller*
CPU	*Central Processing Unit*
CSP	*Cold Spare*
CTMC	*Continuous-time Markov Chains*
DHCP	*Dynamic Host Configuration Protocol*
DNS	*Domain Name System*
DTMC	*Discrete-time Markov Chains*
EBS	*Elastic Block Storage*
EC2	*Elastic Compute Cloud*
ERP	*Enterprise Resource Planning*
FDEP	*Functional Dependency*
ECU	*Elastic Compute Unit*
FTA	*Fault Tree Analysis*
HSP	*Hot Spare*
HTTP	*Hypertext Transfer Protocol*
IaaS	*Infrastructure as a Service*
ICMP	*Internet Control Message Protocol*
IP	*Internet Protocol*
iSCI	*Internet Small Computer System*
ISO	*International Organization for Standardization*
KVM	*Kernel-based Virtual Machine*
MEC	Ministério da Educação e Cultura
MTBF	*Mean Time Between Failures*
MTTF	*Mean Time to Failure*

MTTR *Mean Time to Repair*
NASA *National Aeronautics and Space Administration*
NAT *Network Address Translation*
NC *Node Controller*
NFS *Network File System*
NIST *National Institute of Standards and Technology*
PaaS *Platform as a Service*
PDF *Probability Density Function*
QEMU *Quick Emulator*
QoS *Quality of Service*
RBD *Reliability Block Diagram*
RPC *Remote Procedure Call*
S3 *Simple Storage Service*
SaaS *Software as a Service*
SEQ Sequência Forçada
SISU Sistema de Seleção Unificada
SLA *Service Level Agreement*
SOAP *Simple Object Access Protocol*
SQL *Structure Query Language*
SSH *Secure Shell*
TI Tecnologia da Informação
URL *Uniform Resource Locator*
VLAN *Virtual Local Area Network*
VNC *Virtual Network Computing*
WSP *Warm Spare*
XFS *X File System*
XML *eXtensive Markup Language*

Capítulo 1

Introdução

Atualmente, a área de Tecnologia da Informação (TI) tem apresentado mudanças significativas e rápidas. Muitos trabalhos têm sido sendo realizados sobre a virtualização de *software* e *hardware*. Uma área em especial tem chamado à atenção: Computação em Nuvem. Essa tecnologia tem despertado a atenção do âmbito acadêmico com influência direta no mercado empresarial. Surge então uma nova forma de pensar, de organizar e de gerenciar os recursos tecnológicos como serviços, cujos ambientes podem ser delegados a empresas especializadas em prestação de serviços [Armbrust et al. 2009].

Os usuários e as empresas têm transferido seus dados e suas aplicações para a nuvem, para que possam acessar de maneira mais simples, necessitando somente de uma conexão com a *Internet*. Assim sendo, as empresas e os usuários não precisam de se preocupar em como as aplicações foram implantadas nem onde estão instaladas. Essas aplicações podem variar significativamente em termos de necessidade de recursos, tipo de acesso e dependências.

O termo Computação em Nuvem é decorrente do local em que a computação é feita, isto é, externamente ao local de trabalho, em algum local físico distante, na maioria das vezes desconhecido e onde seu acesso é disponível via *Internet*. O *National Institute of Standards and Technology (NIST)*, define Computação em Nuvem como uma tecnologia que permite gerenciar recursos compartilhados tais como: servidores, redes, sistemas de armazenamento e serviços. É possível, então, seu fornecimento de forma rápida, com mínimo de esforço gerencial ou interação com o prestador de serviços [Jansen e Grance 2011]. As características essenciais para a Computação em Nuvem são serviço sob demanda, acesso em banda larga, *pooling* de recursos, elasticidade rápida e bilhetagem [Dillon et al. 2010].

Atualmente, as empresas deixam de se preocupar com a compra de super computadores e investem em mobilidade, em portabilidade e em elasticidade. Todas essas características estão presentes na Computação em Nuvem. Usar a Computação em Nuvem significa usar uma solução que já nasceu dinâmica, que permite que as soluções escalem, sejam elas quais forem, com custos proporcionais à utilização dos recursos.

A Computação em Nuvem foi desenvolvida com o objetivo de fornecer serviços com facilidade de acesso e com preços reduzidos, garantindo a todos os usuários os atributos da dependabilidade de sistemas. A Computação em Nuvem visa a fornecer três benefícios aos usuários [Sousa et al. 2009]:

- **Redução nos Custos da Infraestrutura:** não é necessário mais investir em super computadores. Com a Computação em Nuvem, a infraestrutura para atender às necessidades de uma empresa pode ser adquirida sob demanda, definindo os principais recursos necessários para cada tipo de empresa, que pagará somente por aquilo que utilizou;
- **Flexibilidade:** a Computação em Nuvem permite a inserção, a alteração ou a remoção da configuração de recursos computacionais de maneira rápida e fácil. Atende melhor, assim, às necessidade dos usuários, pois apresenta alta escalabilidade nos recursos de *hardware* e *software*;
- **Fácil Acesso:** os recursos de TI são fornecidos como um serviço, os usuários acessam os serviços sem precisar conhecer sobre a tecnologia usada. Todos os serviços são providos aos usuários de forma transparente. Para utilizar os serviços é necessário somente uma conexão com a *Internet*, podendo acessar de qualquer lugar e utilizando diversos dispositivos.

Um dos maiores desafios aos provedores de nuvem é manter o controle sobre a qualidade dos serviços prestados, do uso eficiente dos recursos computacionais. Logo, o que foi acordado no contrato de prestação de serviço entre o usuário e o provedor deve ser cumprido. Para assinatura deste contrato utiliza-se o Acordo em Nível de Serviço ou *Service Level Agreements*(SLA), que delimita de forma clara a responsabilidade de cada parte e também traz a definição dos requisitos que estão sendo contratados, para que, no futuro, seja possível avaliar o cumprimento dos contratos e a eficiência dos recursos.

Devido a essa grande tendência pelos serviços de Computação em Nuvem, percebe-se a necessidade de uma arquitetura que garanta níveis confiáveis de dependabilidade. A dependabilidade é a capacidade dos sistemas computacionais de prover, entregar ou prestar um serviço que se pode justificadamente confiar [Avizienis et al. 2004]. Portanto, é a confiança depositada no sistema computacional em relação ao seu correto funcionamento. Definir e modelar a dependabilidade na Computação em Nuvem não é uma tarefa trivial, devido às suas características de compartilhamento de serviços em grande escala, ampla área de rede, componentes de *software* e *hardware* heterogêneos e interações complexas entre eles. Diversos problemas de dependabilidade já ocorreram em ambientes de Computação em Nuvem. Têm-se alguns exemplos:

1. Diversos alunos perderam a inscrição para a prova do Sistema de Seleção Unificada (SISU) do Ministério da Educação e Cultura (MEC) no dia 16/01/2011, devido a um problema ocorrido no site do SISU [Educação 2011];

2. A emissão de passaportes no País foi prejudicada em 19/09/2011, devido a uma falha no sistema da Polícia Federal [S.Paulo 2011];
3. Falhas nos servidores da *Amazon* em abril de 2011, acarretaram a retirada do ar diversos serviços de redes sociais, como por exemplo, o *Foursquare* e o *Quora* [de São Paulo 2011];
4. O roubo de informações dos usuários do *Play Station Network*, em abril de 2011. Os dados de mais 24 milhões de usuários foram roubados. Essa invasão levou cerca de sete dias para ser descoberta, deixando evidente a vulnerabilidade de segurança do sistema pela capacidade do invasor em explorar as falhas existentes [TecMundo 2011].

Recentemente um atributo da dependabilidade, mais especificadamente a confidencialidade das informações foi discutido pela mídia. A mídia nacional e internacional trouxe informações sobre os programas de espionagem realizados pelo governo americano em vários países ao redor do mundo. Os americanos detêm de um forte esquema de espionagem para capturar informações privilegiadas de grandes empresas e de políticos de diversos países, utilizando informações de servidores de grandes empresas como Google e Facebook [InfoEscola 2013] [Terra 2013]. O esquema de espionagem realizado pelo governo americano representa uma violação aos direitos humanos dos cidadãos e contra a soberania dos países que foram espionados.

Esses fatos relatados pela mídia mostram a necessidade de prover um ambiente funcional, confiável e seguro utilizando os conceitos de dependabilidade. O desenvolvimento desta Dissertação de Mestrado se deu pela necessidade de uma arquitetura de serviço para o monitoramento automatizado dos recursos computacionais, que afetam a dependabilidade na Computação em Nuvem. Essa é uma tarefa complexa e que necessita de um alto grau de abstração, tendo em vista a quantidade de recursos totalmente heterogêneos disponíveis nesse tipo de ambiente computacional.

O objetivo principal desta Dissertação de Mestrado é especificar e implementar uma arquitetura de Computação em Nuvem, denominada ArqDep, que possibilite o monitoramento de diversos recursos que afetam a dependabilidade online. Alguns elementos que são utilizados pela arquitetura de referência do NIST [Liu et al. 2011] e do *OpenStack* estão presentes na ArqDep. A ArqDep possibilitará o monitoramento em tempo real dos atributos da dependabilidade, em especial, os atributos de confiabilidade e disponibilidade. Assim sendo, todos os usuários da infraestrutura terão a possibilidade de acompanhar o desempenho da infraestrutura durante o fornecimento dos serviços, por meio das métricas que afetam o desempenho, a eficiência, a disponibilidade e a confiabilidade. Essas métricas são disponíveis na infraestrutura para que os usuários possam analisar o comportamento do sistema durante toda a sua vida útil. Para atingir este objetivo principal, foram traçados alguns objetivos específicos.

O primeiro objetivo específico foi idealizar e implantar um modelo de Infraestrutura como Serviço (IaaS); a construção desse ambiente é necessária para realizar a prova de conceito da ArqDep. Essa infraestrutura foi implantada em uma nuvem privada, dentro da Universidade Federal de Uberlândia.

O segundo objetivo foi criar um módulo de gerenciamento, cuja responsabilidade é monitorar e controlar a infraestrutura. Por meio do monitoramento, ele é capaz de entregar diversos relatórios específicos aos usuários e aos administradores da infraestrutura. Mediante controle, o módulo tem como responsabilidade classificar as falhas apresentadas na infraestrutura e realizar a realocação de recursos entre as máquinas virtuais.

O terceiro objetivo foi realizar a modelagem da dependabilidade do sistema utilizando o modelo Árvore de Falhas. Existe uma busca por confiabilidade e disponibilidade não só em sistemas críticos, mas também em sistemas de uso pessoal e empresarial. A Computação em Nuvem é um paradigma complexo e heterogêneo que busca essa confiabilidade e essa disponibilidade. Prover um serviço livre de falhas, e disponível assim que solicitado, é um dos grandes desafios dos vários provedores de serviços em nuvem [de Araújo Macêdo et al. 2010].

Diversas empresas de médio e grande porte necessitam de um ambiente em Computação em Nuvem com atributos de confiabilidade e disponibilidade dos serviços, que estejam de acordo com o contrato estabelecido entre o provedor e o consumidor. Com isso, a ArqDep atende à demanda dessas empresas, pois a Infraestrutura como Serviço, implantada na nuvem privada, terá o monitoramento contínuo de diversos recursos computacionais que afetam a confiabilidade e a disponibilidade dos sistemas. Modelar e monitorar a confiabilidade e a disponibilidade do sistema não é uma tarefa trivial, mas pode trazer diversas melhorias aos sistemas de Computação em Nuvem.

Muitas empresas implementam suas próprias infraestruturas de Computação em Nuvem, para não dependerem dos diversos provedores desses serviços. A arquitetura proposta nesta Dissertação tem como objetivo auxiliar esse nicho de mercado, pois utilizando essa arquitetura, as empresas vão implantar um ambiente que apresenta um gerenciamento completo da infraestrutura. Logo, os administradores da infraestrutura obterão diversas métricas que os auxiliarão na tomada de decisões para garantir a confiança do sistema.

Aos usuários finais da infraestrutura pode-se destacar a avaliação em tempo real dos diversos recursos computacionais que afetam a disponibilidade e a confiabilidade do sistema. Os dados coletados durante o monitoramento da infraestrutura são disponibilizados a esses usuários. Logo, eles podem verificar se o que foi acordado no SLA está realmente sendo entregue corretamente. Diversas atividades, como auditoria de segurança, detecção de falhas, manutenção preventiva e corretiva, escalonamento de trabalho, desempenho dos recursos e muitas outras só são possíveis de serem realizadas por meio da coleta constate de informações do sistema. Sendo assim, a ArqDep atenderá a todas as atividades citadas anteriormente.

Diversos cenários de testes podem ser utilizados para comprovar a eficiência dessa arquitetura. Para demonstrar a eficiência da infraestrutura, foram reproduzidas situações do dia a dia dos usuários de serviços de Computação em Nuvem. Foram criadas diversas instâncias de máquinas virtuais na infraestrutura, a partir de diferentes imagens. Essas instâncias foram monitoradas e coletaram-se métricas de diversos recursos computacionais que afetam a disponibilidade e confiabilidade dos sistemas.

O Capítulo segundo contempla uma revisão da literatura sobre os conceitos básicos de Computação em Nuvem. No primeiro momento, são descritas as principais definições e as características essenciais da Computação em Nuvem; esses conceitos são essenciais para a compreensão da presente Dissertação. Uma revisão acerca dos modelos de serviços da Computação em Nuvem é realizado. Os modelos de implantação de nuvem também são descritos. Posteriormente, detalhes sobre o SLA são apresentados como uma forma de o leitor compreender como funcionam os contratos assinados entre o provedor de serviço em nuvem e os consumidores. E por fim, as principais arquiteturas de provedores de Computação em Nuvem são detalhadas, a do *Amazon Elastic Compute Cloud (Amazon EC2)* e a *Rackspace*.

O Capítulo terceiro apresenta as plataformas de gerenciamento de nuvem. Inicialmente a plataforma *Elastic Utility Computing Architecture Linking Your Programs To Useful Systems (Eucalyptus)* é descrita. Os componentes da plataforma são apresentados e sua arquitetura é detalhada. Posteriormente é descrito a plataforma *OpenStack* na versão *Havana* com os seus componentes básicos.

Em seguida, é realizada uma revisão sobre a dependabilidade. Os principais conceitos a cerca da dependabilidade são discutidos, visto que esses conceitos são fundamentais para essa Dissertação de Mestrado. Inicialmente destacam-se os atributos da dependabilidade, as ameaças a dependabilidade de sistemas computacionais e os meios para alcançar a dependabilidade. Na sequência do capítulo, são apresentados os diversos modelos de confiabilidade, detalhando as características de cada modelo.

O Capítulo quarto descreve a arquitetura proposta, chamada de ArqDep. No primeiro momento, uma visão geral sobre a arquitetura é abordada. Em seguida, é detalhada a relação entre os atores da arquitetura mostrando a dependência de cada ator dentro do sistema, bem como os módulos e suas relações são detalhados. O *hypervisor* utilizado foi o *Quick Emulator (QEMU)*, pois ele pode ser utilizado em *hardware* que não suportem virtualização pelo processador que é o nosso caso. No decorrer do capítulo é feita a implementação da proposta. O módulo de gerenciamento, responsável pelo monitoramento e controle da infraestrutura é delineado e também é realizada uma avaliação da dependabilidade do sistema utilizando o modelo de Árvore de Falhas. Por último os resultados obtidos são descritos.

As conclusões desta Dissertação de Mestrado são apresentadas no Capítulo quinto, juntamente com as sugestões de trabalhos futuros. Também são ressaltados os pontos

positivos e negativos encontrados durante o desenvolvimento desta pesquisa.

Capítulo 2

Computação em Nuvem

Neste capítulo, são apresentados os conceitos que fundamentam a Computação em Nuvem. Esses aspectos conceituais são importantes para compreender seus paradigmas. São discutidos os modelos de serviços utilizados para prover os serviços de Computação em Nuvem de acordo com o NIST e os tipos de modelos usados para a implantação da nuvem que podem ser privada, pública, híbrida ou comunidade. O modelo de serviço e o tipo de implantação da nuvem são determinados de acordo com os requisitos levantados para o projeto. Cada projeto pode apresentar requisitos particulares, que demandam arquiteturas diferentes. No decorrer do capítulo, são descritas duas arquiteturas do modelo de serviço IaaS de nuvem pública disponíveis no mercado atualmente. Todos os provedores de serviços em nuvem, no ato do fechamento dos contratos com os seus consumidores de serviços, assinam um SLA, no qual estão descritos os direitos e deveres de ambas as partes. Ao final do capítulo, são abordados detalhes de como funciona o SLA.

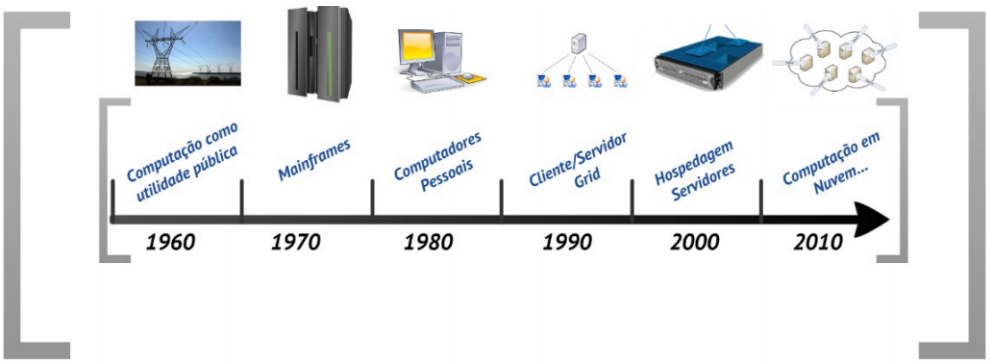
2.1 Conceitos Básicos

A Computação em Nuvem é uma tecnologia que permite novos modelos de negócios, em que os recursos computacionais tais como processamento, armazenamento, rede e *software* são oferecidos pela *Internet* e podem ser acessadas remotamente [Armbrust et al. 2009]. As informações ficam disponíveis aos usuários e eles podem acessá-las de qualquer lugar do mundo. Cada componente de uma nuvem computacional é disponibilizado como um serviço e eles são normalmente alocados em *data centers*, utilizando *hardware* compartilhado para computação e armazenamento.

Os computadores dos usuários dos serviços de Computação em Nuvem não necessitam de configurações avançadas de *hardware*, diminuindo assim, o custo envolvido na aquisição e manutenção de equipamentos computacionais [Sousa et al. 2009]. A Computação em Nuvem emergiu como uma das mais influentes tecnologias na indústria de TI e está revolucionando o modo pelo qual os recursos de TI são gerenciados e utilizados [Armbrust et al. 2009].

A linha do tempo apresentada no Figura 2.1 mostra a evolução da tecnologia até chegar a Computação em Nuvem. Em 1960, por incrível que pareça, Joseph Carl Robnett já pensou na ideia de utilização de recursos computacionais por meio da *Internet*. Ele foi um dos desenvolvedores da ARPANET, o antecessor direto da *Internet*, porque já imaginava computadores interligados por uma rede, em que todos estariam conectados acessando os programas em qualquer lugar. Já John McCarthy, definiu a computação nesta época como uma utilidade pública.

Figura 2.1: Linha do tempo da Computação em Nuvem



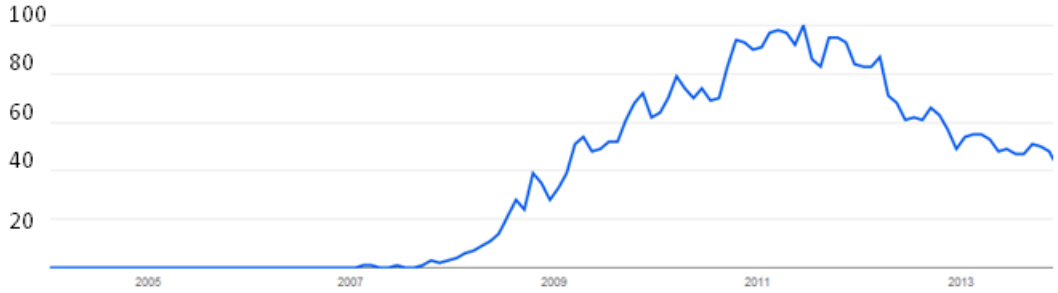
Fonte: Autora

Em 1970 foi a vez dos *Mainframes*, época dos grandes computadores, que ocupavam na maioria das vezes, uma sala inteira no escritório. A empresa líder de mercado nessa época era a IBM. Em seguida, o mercado adotou o uso de computadores pessoais. Em 1990, foi a vez da arquitetura Cliente/Servidor, em que os usuários começaram a compartilhar suas informações por intermédio dos servidores. No ano de 2000, foi a vez da hospedagem em servidores.

A primeira vez que se utilizou o termo Computação em Nuvem foi em 1997. Ele foi utilizado pelo professor Ramnath Chellappa em uma palestra acadêmica. Em 1999 surgiu, a Salesforce.com, empresa pioneira na disponibilização de aplicações pela *Internet*. A partir daí, diversas empresas investiram na área. A Computação em Nuvem mudou o modelo de negócio na área de TI e está em crescente crescimento. A Figura 2.2 ilustra o crescimento em pesquisas globais sobre o termo *Cloud Computing* [Google 2014].

Os números presentes no gráfico indicam quantas pesquisas foram realizadas para o termo *Cloud Computing*, em relação ao número total de pesquisas feitas no Google ao longo do tempo. Eles não representam valores absolutos de volume de pesquisa, pois os dados são normalizados e apresentados em uma escala de 0 a 100. Nota-se que a busca pelo termo *Cloud Computing* iniciou-se entre os anos de 2007 e 2008. Entre 2011 e 2012, a busca pelo termo atingiu o ponto máximo.

Figura 2.2: Pesquisas globais sobre o termo *Cloud Computing*



Fonte: Google Trends

Diversos produtos lançados só surgiram no mercado devido à Computação em Nuvem. Por exemplo, os serviços de armazenamento remoto de arquivos como o Dropbox e Google Drive. Esses serviços estão disponíveis aos usuários em virtude da Computação em Nuvem. Ao utilizar esses serviços, os usuários acessam seus arquivos em qualquer dispositivo que tenha acesso à *Internet*, basta ter uma conta associada ao serviço.

2.1.1 Definições

Não existe uma definição consensual sobre o paradigma de Computação em Nuvem na literatura. Diversas definições de Computação em Nuvem são encontradas na literatura por vários autores. A definição mas comumente usada é a definição proposta pelo NIST [Mell e Grance 2011]. O NIST define Computação em Nuvem como um modelo computacional que permite o acesso ubíquo, conveniente, sob demanda e mediante a rede a um *pool* de recursos computacionais configuráveis, tais como, redes, servidores virtuais ou físicos, armazenamento, aplicações e serviços, que podem ser rapidamente provisionados e liberados.

A Computação em Nuvem também é definida como um conjunto de serviços de rede ativados, proporcionando escalabilidade, qualidade de serviço, infraestrutura barata de computação sob demanda e que pode ser acessada de uma forma simples, utilizando a *Internet* por meio de um *browser* e pervasiva [Armbrust et al. 2009]. Nessa definição, destaca-se a escalabilidade, a diminuição de custos com a infraestrutura, pois não é necessário altos investimentos com *hardware* e *software*, e o modo pelo qual os serviços prestados na nuvem são acessados facilmente.

A Computação em Nuvem pode ainda ser definida como um estilo de computação no qual recursos de TI escaláveis e elásticos são providos como serviços para consumidores usando tecnologias de *Internet* e são bilhetados pelo uso [Cearley e Phifer 2009]. Além disso, a Computação em Nuvem pode ser vista como um paradigma de computação distribuída em larga escala, no qual um *pool* de recursos abstratos, virtualizados, escaláveis,

gerenciados, plataformas ou serviços, que são entregues sob demanda a consumidores externos por meio da *Internet* e são cobrados pela sua utilização [Foster et al. 2008]. As duas últimas definições trazem palavras semelhantes na definição de Computação em Nuvem, onde se devem destacar as seguintes palavras:

- **Escaláveis:** Os recursos de Computação em Nuvem são escaláveis de forma transparente aos usuários.
- **Serviços:** A forma de cobrança dos serviços de Computação em Nuvem é similar ao fornecimento de serviços utilitários como energia e gás, o usuário paga somente pelo que utilizar.
- ***Internet*:** O meio que os usuários utilizam para terem acesso aos serviços de Computação em Nuvem fornecidos pelos provedores.

A Computação em Nuvem não é um conceito único e, sim, um termo abstrato, que envolve diversos conceitos e tecnologias. Diferentes provedores de nuvem oferecem vários serviços e soluções para atenderem a demanda por serviços de Computação em Nuvem [Fouquet et al. 2009]. Os provedores de serviços em nuvem existentes no mercado oferecem serviços como: hospedagem de site, processamento, plataformas de desenvolvimento, hospedagem de aplicativos corporativos, banco de dados e diversos outros serviços.

Logo, a Computação em Nuvem está associada a um novo paradigma de fornecimento de infraestrutura para a computação, em que a localização da infraestrutura é transferida para a rede, reduzindo custos associados a *hardware* e *software* [Vaquero et al. 2008]. Os usuários dos serviços não precisam conhecer a localização física da infraestrutura, pois os serviços são acessados de qualquer lugar utilizando a *Internet*.

De forma geral, a Computação em Nuvem possui algumas palavras-chave como: serviços, sob demanda, escalabilidade e elasticidade. Essas palavras estão relacionadas com as características essenciais da Computação em Nuvem que veremos a seguir. A escalabilidade e elasticidade são relacionadas à percepção de recursos ilimitados. Os serviços são sob demanda, similares ao fornecimento de serviços utilitários como gás e energia elétrica, onde são contabilizados somente os serviços utilizados.

2.1.2 Características Principais da Computação em Nuvem

Para um melhor entendimento sobre a Computação em Nuvem, é necessário conhecer suas principais características: serviço sob demanda, acesso em banda larga, *pooling* de recursos, elasticidade rápida e bilhetagem, similar às do sistema de distribuição elétrica [Mell e Grance 2011]. Essas características são descritas sucintamente a seguir:

- **Serviço sob demanda:** É a capacidade de um consumidor provisionar recursos computacionais tais como servidores e memórias sem a necessidade de intervenção

humana. Por exemplo, o consumidor pode solicitar uma nova instância de máquina virtual, de forma automática quando surgir a necessidade, sem nenhuma intervenção de uma pessoa do lado provedor de serviços. O consumidor pode alterar as configurações dos seus recursos sempre que julgar necessário. Todas essas modificações devem ser realizadas sem nenhuma intervenção humana por parte do provedor. Essa automatização é possível quando ferramentas de gerenciamento são disponibilizadas pelo provedor de serviços aos consumidores.

- **Acesso em banda larga:** É a capacidade de disponibilizar os recursos e serviços Por meio das redes por qualquer dispositivo como *pdas*, *notebooks* e celulares. Os serviços podem ser acessados por meio de mecanismos padronizados que promovam o uso de plataformas clientes heterogêneas, basta somente um dispositivo com acesso a *Internet* para ter acesso aos serviços fornecidos pelo provedor.
- ***Pooling* de recursos:** É a capacidade de atender a vários consumidores ou inquilinos por meio de arquiteturas como *MultiTenant*, que se referem a um princípio da arquitetura de *software*, em que uma única instância do *software* é executada em um servidor para atender às requisições de múltiplos clientes *(tenants)*. A arquitetura *MultiTenant* permite que múltiplos *tenants* compartilhem os mesmos recursos físicos, como, por exemplo, um aplicativo *Enterprise Resource Planning (ERP)*, mas permaneçam logicamente isolados [Taurion 2009]. Os modelos *MultiTenants* são:
  - ***Tenant isolado:*** Cada *tenant* apresenta a sua pilha de tecnologia, não há nenhum compartilhamento de recursos. De uma forma prática, o usuário tem a sensação de estar utilizando o *MultiTenant*, pois a aplicação é oferecida a múltiplos clientes a partir do mesmo *data center*. Esse modelo é similar ao modelo tradicional de hospedagem, em que cada usuário tem o seu próprio conjunto de recursos computacionais e sua própria instância da aplicação.
  - ***MultiTenant via hardware compartilhado:*** Neste modelo cada *tenant* possui a sua própria pilha de tecnologia, entretanto o *hardware* é alocado dinamicamente a partir de um *pool* de recursos, por meio da virtualização. Esse modelo permite elasticidade na camada do *hardware*.
  - ***MultiTenant via container:*** Vários *tenants* são executados em uma mesma instância de um *container* de aplicação, porém cada *tenant* está associado a uma instância separada do *software* de banco de dados.
  - ***MultiTenat via todo o stack de software compartilhado:*** É uma variação do modelo anterior, no qual toda a pilha de *software* é compartilhada. Além do *container* da aplicação, a instância do banco de dados também é compartilhada por todos os *tenants*.

Os recursos computacionais disponíveis pelos provedores devem estar agrupados

para servir a vários consumidores ao mesmo tempo. Esses recursos são alocados, desalocados e realocados dinamicamente, conforme a demanda. O consumidor dos recursos não tem conhecimento ou controle da localização exata dos recursos fornecidos pelo provedor.

- **Elasticidade rápida:** Os recursos podem ser rápida e elasticamente provisionados, tanto quanto para aumentar a capacidade, quanto para liberar os recursos computacionais. Logo, os provedores de nuvem fornecem e liberam os recursos computacionais de acordo com a demanda do usuário. Todas essas tarefas são feitas de forma automática, dando a sensação ao consumidor de recursos infinitos, que podem ser requisitados a qualquer momento. A virtualização é responsável por essa elasticidade de recursos.
- **Bilhetagem:** É a realização do monitoramento, controle e emissão de relatórios de cobranças, provendo assim mais transparência para ambos provedores e consumidores. Similar ao pagamento do consumo de energia elétrica, o usuário só paga pelo que utilizou. Cada provedor de serviços de Computação em Nuvem pode adotar uma forma diferente de cobrança, por exemplo, os recursos de processamento podem ser cobrados por hora e os serviços de armazenamento por mês ou quantidade de dados armazenados.

Essas características principais da Computação em Nuvem são providas em diferentes modelos de serviços, sendo de interesse dos usuários os recursos que são providos. Cada provedor adota suas características ao projetar a sua infraestrutura. Um aspecto importante a se discutir sobre a Computação em Nuvem é como as infraestruturas são projetadas. Ao iniciar um projeto de construção de uma infraestrutura de Computação em Nuvem é necessário definir que serviços aquela infraestrutura oferecerá aos seus consumidores. Após essa definição, é necessário escolher o modelo de serviço a ser utilizado e o modelo de implantação de nuvem para implementar a infraestrutura.

2.2 Modelos de Serviços

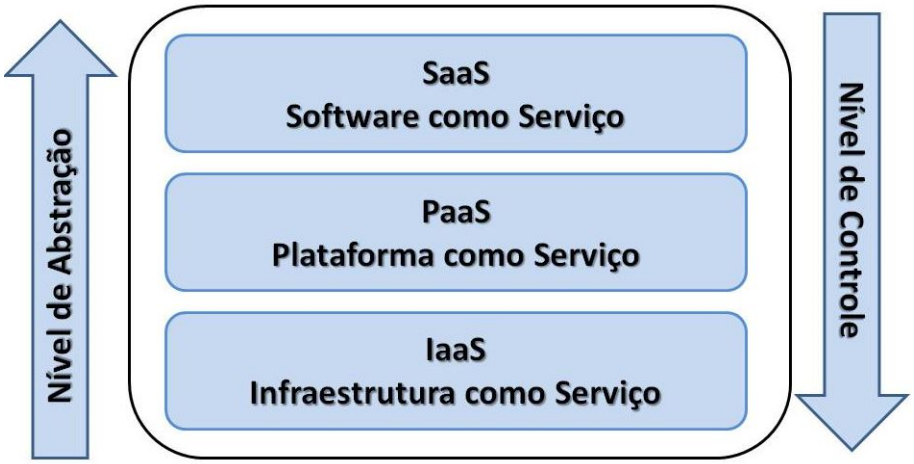
Os serviços oferecidos na Computação em Nuvem envolvem plataforma de *software* e *hardware* sob demanda. Por meio da virtualização, pode-se criar um ambiente escalável, em que os usuários podem obter os recursos computacionais de que necessitam, sem a necessidade de interação humana na infraestrutura. O *software* e *hardware* dentro da nuvem podem ser reconfigurados de forma automática, orquestrados e essas modificações são transparentes aos usuários.

Diversos privilégios podem ser configurados para diferentes usuários, assim, é possível personalizar os ambientes computacionais de diferentes clientes. A localização da infraes-

trutura não é de conhecimento dos usuários, alguns provedores disponibilizam somente a região onde se encontra a infraestrutura.

Há três modelos de serviços reconhecidos na Computação em Nuvem pelo NIST [Jansen e Grance 2011]. Esses modelos são importantes, pois eles definem um padrão da arquitetura para soluções de Computação em Nuvem. Os três modelos de serviços são: *Software* como Serviço, Plataforma como Serviço e Infraestrutura como Serviço. Cada modelo tem o seu nível de abstração e controle conforme mostra a Figura 2.3.

Figura 2.3: Modelos de Serviços



Fonte: [Jansen e Grance 2011]

A Figura 2.3 apresenta em camada os modelos de serviços com o nível de abstração e o nível de controle de cada modelo. O nível de abstração está relacionado com a não responsabilidade sobre os componentes da infraestrutura. O nível de controle está relacionado ao gerenciamento do modelo. Quanto mais alta a camada, maior o nível de abstração e menor o controle. Por exemplo, ao contratar um serviço SaaS, o usuário não tem envolvimento com o desenvolvimento, gerenciamento ou administração da solução. Quanto mais baixa a camada do modelo de serviço, maior será o controle e menor a abstração. Como exemplo, quando um usuário contrata um serviço IaaS, ele contrata uma infraestrutura capaz de executar e hospedar seu *software* sobre o sistema operacional que escolheu, instalar, administrar e gerenciar a sua utilização. A seguir a definição de cada modelo apresentado na Figura 2.3.

2.2.1 *Software* como um Serviço (SaaS)

O modelo de serviço SaaS é um *software* oferecido em forma de serviço ou prestação de serviço. O consumidor contrata a utilização de uma aplicação que está hospedada e executada na nuvem. Não é necessário instalar o sistema no computador do cliente, basta acessá-lo pela *Internet* com um *browser*. O SaaS é um conceito atrativo para todos os profissionais de TI e usuários comuns em que se deparam, a cada dia, com

software com infindáveis atualizações, correções e alto custo de licenças. Nesse modelo, os aplicativos são executados no ambiente da nuvem, ficando disponíveis por meio da *Internet* e acessíveis de qualquer lugar a partir de diversos dispositivos conectados à *Internet*.

No SaaS, os usuários não administram ou controlam a infraestrutura, como por exemplo, rede, servidores, sistemas operacionais e armazenamento. O responsável por administrar essa infraestrutura, pelo desenvolvimento e atualizações das aplicações fornecidas aos usuários é o provedor da nuvem. Esse modelo de serviço está cada vez mais presente no dia a dia de diversos usuários. Exemplos de tipos de serviços são *Google Docs* [Ciurana 2009] e *Sales Force* [Salesforce 2013].

2.2.2 Plataforma como um Serviço (PaaS)

O modelo de serviço PaaS é a versão intermediária da Computação em Nuvem. Oferece uma infraestrutura de alto nível de integração para implementar e testar aplicações na nuvem. O usuário não administra ou controla a infraestrutura, mas tem controle sobre as aplicações implantadas. O PaaS fornece um sistema operacional, linguagens de programação e ambientes de desenvolvimento para as aplicações, auxiliando a implementação de sistemas de *software*. No PaaS tem-se o provisionamento de serviços que permitem o desenvolvimento, testagem, implantação, hospedagem e gerenciamento de aplicações com o objetivo de suportar o ciclo de vida de desenvolvimento de aplicações.

O modelo PaaS fornece ambientes de desenvolvimento de *software*, facilitando a implantação de aplicativos sem os custos e complexidades relativas a compra e gerenciamento de *hardware* necessários para o desenvolvimento de aplicações. Diversos serviços podem ser oferecidos por meio desse modelo de serviço para facilitar projetos de aplicativos. Exemplos de serviços são *Google App Engine* [Ciurana 2009] e *Aneka* [Vecchiola et al. 2009].

2.2.3 Infraestrutura como Serviço (IaaS)

O modelo de serviço IaaS é o responsável de prover toda a infraestrutura necessária para o PaaS e o SaaS. Tem como principal objetivo fornecer uma infraestrutura de processamento, armazenamento, servidores e rede como serviço. Nesse modelo, o consumidor do serviço é cobrado conforme o uso dos recursos. No IaaS o usuário não administra ou controla a infraestrutura em nuvem, mas tem controle sobre os sistemas operacionais, armazenamento e aplicativos implantados.

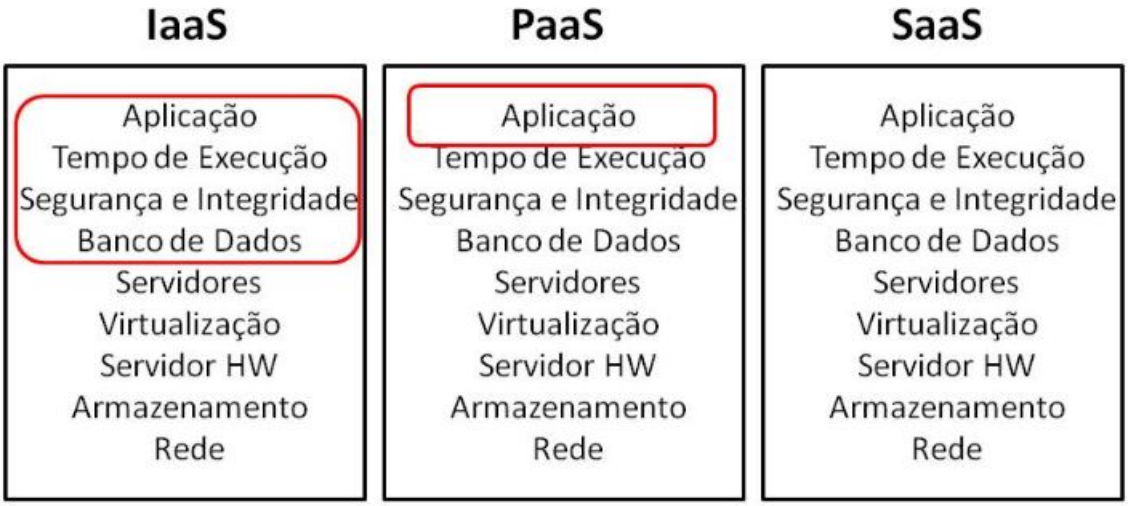
O modelo IaaS oferece uma infraestrutura que pode ser escalada dinamicamente, aumentando ou diminuindo os recursos de acordo com as necessidades das aplicações. Suas características o tornam muito rentável para os usuários, porque em vez de comprar novos servidores e equipamentos de rede para a ampliação de serviços, podem-se aproveitar

os recursos disponíveis e adicionar novos à infraestrutura existente de forma dinâmica e transparente. Várias empresas disponibilizam esses serviços por meio da Internet e de máquinas virtuais, como exemplos: *Amazon Elastic Cloud Computing (EC2)* [Robinson 2008] e *Rackspace* [Rackspace 2013].

2.2.4 Gerenciamento de Serviços

Cada modelo de serviço reconhecido pelo NIST apresenta a suas particularidades em relação à prestação de serviços aos consumidores. O gerenciamento desses serviços também é realizado de formas diferentes em cada modelo. Em um modelo tradicional, uma empresa monta toda a sua infraestrutura de TI, que é responsável por gerenciar toda essa infraestrutura, incluindo *software* e *hardware*. Nos modelos de serviços de Computação em Nuvem, essas responsabilidades, na maioria das vezes, são repassadas aos provedores dos serviços de Computação em Nuvem dependendo do modelo de serviço utilizado. A Figura 2.4 mostra como é realizado o gerenciamento dos modelos de serviços.

Figura 2.4: Gerenciamento de Serviços



Fonte: Autora

Em IaaS, as aplicações, o tempo de execução, a segurança, a integridade e o banco de dados são gerenciados pelo usuário; já os servidores, a virtualização, o *hardware*, o armazenamento e a rede são obrigações dos provedores de serviços. Em PaaS, somente as aplicações são gerenciadas pelo usuários, os demais itens são de responsabilidade do provedor de serviço. Por último tem-se o SaaS em que todos os itens são de responsabilidade do provedor de serviço.

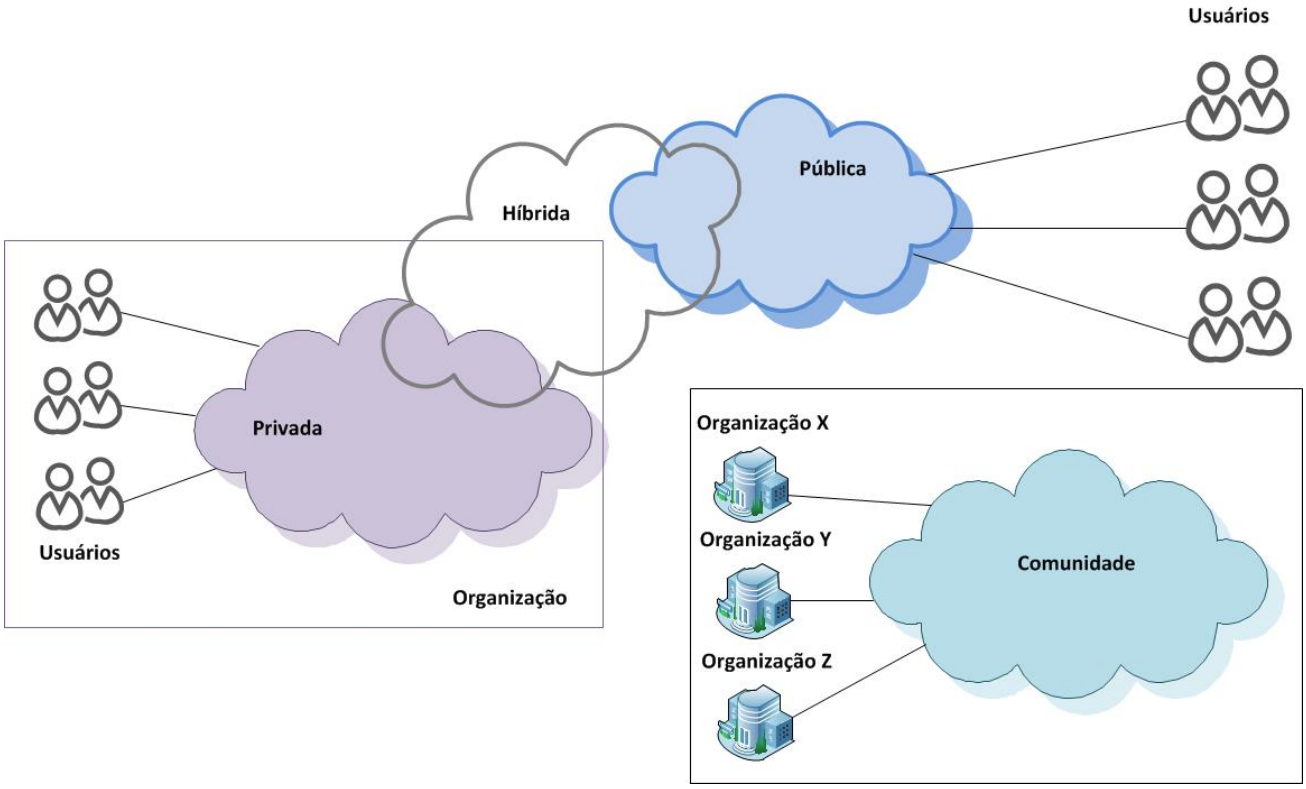
A definição do modelo de implantação da nuvem é um dos fatores importantes durante a elaboração do projeto da infraestrutura. O modelo de implantação é o que estabelece a forma de acesso e de controle aos serviços fornecidos pelos provedores. A próxima seção

apresenta os modelos de implantação de nuvem, que dependem do tipo de acesso aos serviços. Muitas vezes, empresas necessitam de ambientes restritos, em que somente pessoas autorizadas tenham acesso às informações e serviços, sendo necessário um ambiente mais restrito. Em outras ocasiões, o acesso tende a ser liberado para todos os usuários que tenham conhecimento do serviço.

2.3 Modelos de Implantação de Nuvem

A Computação em Nuvem permite diversos tipos de modelos de implantação de Nuvem. Os tipos de Nuvens podem ser descritos quanto à natureza do acesso e do controle em relação ao uso e provisionamento de recursos físicos e virtuais, portanto a restrição ou abertura de acesso depende do processo de negócio, do tipo de informação e do nível de visão. O acesso a um tipo de serviço pode ser limitado dentro de uma organização para cada usuário. Portanto em algumas organizações, o ambiente de Computação em Nuvem necessita de autorização para a utilização dos serviços. Existem quatro modelos de implantação de Computação em Nuvem, como mostra a Figura 2.5 [Mell e Grance 2011].

Figura 2.5: Modelos de implantação de Nuvem



Fonte: Autora

Os quatros modelos ilustrados na Figura 2.5 são utilizados para atender a uma organização ou a pessoas de uma forma diferenciada. Os modelos de implantação de Nuvens,

de uma forma sucinta, realizam uma classificação do público-alvo ou a abrangência de público que utilizará os serviços fornecidos pela infraestrutura. Então, os provedores de serviços em nuvem podem oferecer seus recursos de uma forma livre ou realizar um controle, restringindo acesso aos serviços a um determinado grupo de usuários. Os modelos de implantação de Nuvem serão descritos próximas seções.

2.3.1 Nuvem Privada

No modelo de implantação de Nuvem Privada, toda a infraestrutura e serviços disponíveis na nuvem são disponibilizados para uma única organização, independente dos recursos serem próprios ou terceirizados. Esses recursos não estão disponíveis ao público geral. Toda a infraestrutura é operada apenas por uma organização, podendo ser gerenciada pela própria empresa, ou, eventualmente, por terceiros e pode tanto estar localizada dentro ou fora dos limites físicos da organização. Esse modelo é, na maioria das vezes, construído para adaptar as necessidades de um consumidor específico [Mell e Grance 2011].

O modelo de Nuvem Privada tem como principal objetivo dar aos usuários locais uma infraestrutura ágil e flexível para atender às suas cargas de trabalhos de serviços dentro do seu próprio domínio administrativo, porque esse modelo não quer somente vender recursos pela *Internet*, utilizando interfaces acessíveis ao público de uma forma geral. Dessa maneira, é possível ter um controle mais detalhado dos recursos [Mell e Grance 2011].

2.3.2 Nuvem Pública

O modelo de implantação de Nuvem Pública é disponibilizado para o público em geral; o serviço pode ser acessado por qualquer usuário que tenha conhecimento dele. Os recursos de computação são fornecidos dinamicamente, por meio da *Internet* mediante de aplicações *Web* ou serviços *Web*. Um dos benefícios da Nuvem Pública é que elas podem ser muito maiores do que as privadas, toda a responsabilidade de gerenciamento da infraestrutura é do provedor do serviço [Mell e Grance 2011]. As vantagens que se podem destacar, nesse modelo de implantação, são:

- Facilidade para iniciar um negócio utilizando a nuvem pública, devido ao seu custo e à agilidade do processo. Nesse modelo, os custos de manutenção de *software*, *hardware* e de largura de banda são cobertos pelo provedor de serviços.
- Os serviços providos são escaláveis, fazendo com a nuvem pública transmita aos usuários a sensação de possuir recursos infinitos.
- Os usuários não sofrem desperdício de recursos, pois os recursos utilizados são pagos por meio da bilhetagem.

Já como desvantagens desse modelo de implantação destacam-se:

- Carência de informações sobre a infraestrutura utilizada. Os provedores desses serviços não divulgam aos seus usuários detalhes sobre o *hardware* físico em que as máquinas virtuais são executadas e também não especifica a largura de banda dos enlaces que conectam essas máquinas virtuais.
- O compartilhamento dos recursos entre vários usuários na nuvem pública.
- Em teoria, alguns dados privados dos usuários ficam disponíveis ao provedor da nuvem. Isso pode causar a espionagem de informações confidenciais dos usuários.

Logo, os provedores de serviços em nuvem pública são responsáveis pela instalação, manutenção, gerenciamento e segurança dos serviços fornecidos, enquanto os usuários somente utilizam os recursos de acordo com a sua demanda. Como exemplos de provedores de nuvem pública tem-se a *Amazon EC2* que prover IaaS, o *Google App Engine* com o PaaS e o *Google Apps for Business*, que oferece os serviços no modelo SaaS.

2.3.3 Nuvem Híbrida

O modelo de implantação de Nuvem Híbrida é a combinação dos dois modelos anteriores da nuvem pública com privada, para solucionar as limitações de cada modelo. Nesse modelo de implantação, tanto a empresa fornecedora do serviço quanto o usuário são responsáveis por manter o serviço em funcionamento. Nesse tipo de nuvem, existe uma flexibilidade maior do que na pública e privada. Existem vantagens ao utilizar esse modelo, porque ele oferece elasticidade rápida, baixo custo da nuvem pública e maior segurança e controle dos recursos das nuvens privadas [Mell e Grance 2011].

2.3.4 Nuvem Comunidade

Diversas empresas compartilham a mesma nuvem; essas organizações compartilham dos mesmos interesses como a missão, os requisitos de segurança, flexibilidade e as políticas. Ela pode ser administrada local ou remotamente, por alguma organização ou por terceiros. Uma nuvem comunidade é constituída por um conjunto de usuários de diversas organizações que compartilham aplicações, serviços e recursos computacionais [Mell e Grance 2011].

Os modelos de implantação de Nuvens citados anteriormente são utilizados nas arquiteturas dos provedores de serviços de Computação em Nuvem e nas arquiteturas projetadas pelas próprias organizações que projetam as suas Nuvens Privadas. Cada provedor de Computação em Nuvem cria a sua própria arquitetura e escolhe o modelo de implantação de Nuvem que adotará na sua infraestrutura. Essa escolha é determinada de acordo com o tipo de acesso que cada provedor quer utilizar durante o fornecimento dos seus serviços.

2.4 Arquiteturas de Computação em Nuvem

Existem diversas arquiteturas de Computação em Nuvem encontradas na literatura. Neste capítulo, são descritas duas das principais arquiteturas de IaaS existentes no mercado mundial. As arquiteturas de Computação em Nuvem são baseadas em camadas e cada camada trata de uma particularidade no modo de disponibilização dos seus recursos para as aplicações [Buyya et al. 2009]. Cada arquitetura apresenta as suas próprias características e limitações. A seguir, a descrição de duas arquiteturas de provedores de serviços em nuvem encontradas disponíveis no mercado. Esses dois provedores foram escolhidos pela sua forte participação no mercado atualmente.

2.4.1 A Arquitetura da *Amazon Elastic Compute Cloud (Amazon EC2)*

O *Amazon EC2 (Amazon Elastic Compute Cloud)* é um modelo de nuvem pública IaaS, que fornece uma capacidade de computação redimensionável na nuvem. Oferece uma interface simples e moderna aos usuários, permitindo que eles obtenham e configurem a capacidade com mínimo esforço. O *Amazon EC2* permite que os usuários rapidamente escalonem a capacidade, para mais ou para menos, à medida que os requisitos de computação forem alterados, assim diminuindo o tempo gasto na configuração do servidor. A cobrança dos serviços prestados pelo *Amazon EC2* é realizada por bilhetagem, em que os usuários só pagam pelos recursos utilizados [Amazon 2013].

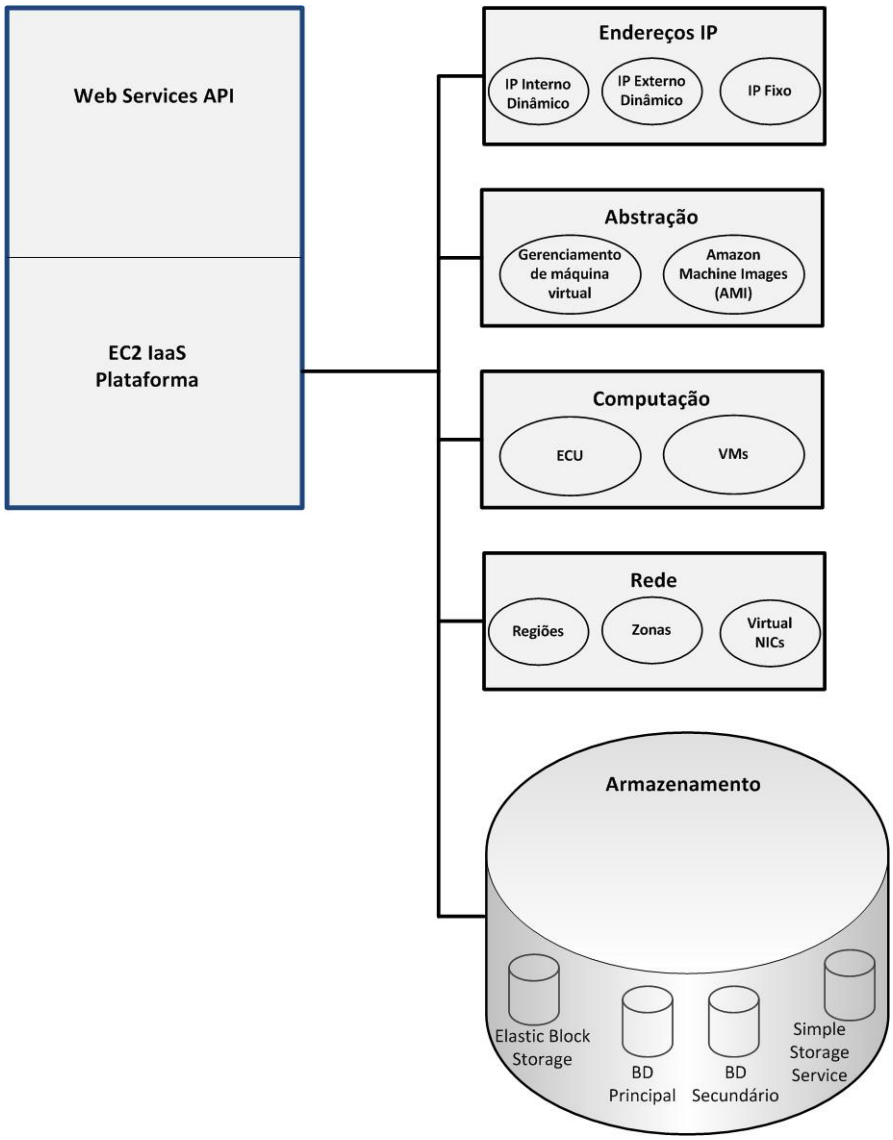
Os principais componentes desta arquitetura são *Elastic Block Storage (EBS)*, *Simple Storage Service(S3)*, Zonas de disponibilidade e regiões, Unidade de processamento EC2 (ECU), Máquinas virtuais, *Software* de virtualização, *Amazon Machine Image (AMI)*, Endereço IP interno dinâmico, Esquemas de endereço IP externo dinâmico e Endereços IP's fixos, conhecidos como *Elastic IP Addresses* [Bojanova e Samba 2011]. A Figura 2.6 mostra a arquitetura básica do Amazon EC2 [Bojanova e Samba 2011].

O *Amazon EC2* utiliza o *hypervisor* Xen para realizar a virtualização; esse serviço permite ao cliente obter e configurar recursos computacionais como processamento, memória e armazenamento, na forma de máquinas virtuais que são executadas em máquinas físicas virtualizadas utilizando o Xen.

O usuário do *EC2* tem a possibilidade de escolher entre oito regiões de disponibilidade para hospedar suas instâncias, que são criadas a partir de instâncias pré-definidas pela *Amazon*. Cada tipo de instância oferece ao usuário uma configuração distinta em relação à capacidade de processamento, armazenamento, memória e desempenho.

A Tabela 2.1 apresenta a relação dos tipos de instâncias oferecidas pela *Amazon* aos seus usuários, com as suas respectivas configurações e preços por hora de utilização para a região da América do Sul, especificamente em São Paulo [Amazon 2013].

Figura 2.6: Arquitetura básica do *Amazon Elastic Compute Cloud*



Fonte: [Bojanova e Samba 2011]

Fonte: [Amazon 2013]

A forma de cobrança do *Amazon EC2* é a bilhetagem, em que o usuário só paga pelos recursos utilizados. O *Amazon EC2* foi projetado para atender aos usuários, em que o tráfego pode chegar a níveis altos e após algumas horas cair bruscamente. Assim sendo, é possível aumentar ou diminuir a capacidade computacional de maneira rápida e simples. O *Amazon* é um dos líderes de mercado em relação à Computação em Nuvem e disponibiliza diversos serviços nesta área.

2.4.2 Arquitetura do *Rackspace*

Fundada em 1998, o provedor *Rackspace* se tornou um líder no fornecimento de serviços em nuvem. O provedor possui um leque de soluções de Computação em Nuvem para

Tabela 2.1: Tipos de instâncias oferecidos pela *Amazon EC2* na região da América do Sul (São Paulo) e suas características e preços por hora de uso em 14 de Outubro de 2013.

Uso do Linux/UNIX				
Instâncias *on demand* padrão	**CPU**	**RAM**	**Armazenamento (GB)**	**Preço**
Pequena (padrão)	1	1,7 GB	160	$ 0.080
Médio	2	3,75 GB	410	$ 0.160
Grande	4	7,5 GB	850	$ 0.320
Extragrande	8	15 GB	1.690	$ 0.640
Instâncias *on demand* padrão de segunda geração				
Extragrande	13	15 GB	EBS	$ 0.680
Dupla extragrande	26	30 GB	EBS	$ 1.360
Micro instâncias *on demand*				
Micro	2	613 MB	EBS	$ 0.027
Instâncias *on demand* com mais memória				
Extragrande	6,5	17,1 GB	420	$ 0.540
Dupla extragrande	13	34,2 GB	850	$ 1.080
Quádrupla extragrande	26	68,4 GB	1.690	$ 2.160
Instâncias *on demand* com CPU de alta performance				
Médio	5	1,7 GB	350	$ 0.200
Extragrande	20	7 GB	1.690	$ 0.800

oferecer a seus clientes. A maioria dessas soluções são no modelo de IaaS. Alguns dos serviços oferecidos pelo Rackspace são [Rackspace 2013]:

- ***Cloud Servers:*** São máquinas virtuais que funcionam com um servidor Linux ou Windows na *Cloud Rackspace*. Os recursos como CPU, memória e armazenamento podem ser redimensionados de acordo com suas necessidades. O pagamento é feito da forma de bilhetagem, o consumidor só paga pelo que ele consumiu. Os preços por hora de uso e as configurações das máquinas virtuais da *Rackspace* são apresentados na Tabela 2.2.

  Fonte: [Rackspace 2013]

  O acesso às máquinas virtuais pode ser feito via *Internet*, utilizando um cliente *Secure Shell (SSH)* ou através de um console de monitoramento disponibilizada pelo próprio provedor.

- ***Cloud Sites:*** É uma plataforma de hospedagem de site e blog gerenciada. O cliente deste serviço não precisa preocupar com *hardware*, rede, sistema operacional,

Tabela 2.2: Configurações e preços de *Cloud Servers* em 07 de Agosto de 2013

Linux			
RAM	**Espaço em disco**	**VCPUs**	**Preço/hora**
512 MB	20 GB	1	$0.022
1 GB	40 GB	1	$0.06
2 GB	80 GB	2	$0.12
4 GB	160 GB	2	$0.24
8 GB	320 GB	4	$0.48
15 GB	620 GB	6	$0.90
30 GB	1,2 TB	8	$1.20

armazenamento, banco de dados, servidores Web, Sistema de Nomes de Domínios (DNS), *firewall*, *cluster*, redundância e *load balancing*. Tudo é de responsabilidade do provedor de nuvem; o que o usuário gerencia são suas aplicações, seus dados, código e a segurança do site. O plano básico desse serviço está no valor de 150 dólares mensais.

- **Servidores dedicados gerenciados:** Utilizado em aplicações e site de alto desempenho, na *Rackspace* os servidores dedicados incluem dispositivo de *firewall*, monitoramento, *backup* e um nível de *Managed Service*. Existem três tipos de servidores dedicados gerenciados: o Servidor *Standard*, o Avançado e o de Desempenho. O preço para adquirir esse serviço é a partir de 499 dólares mensais.
- ***Cloud Files:*** É uma solução de armazenamento que utiliza a tecnologia do *OpenStack*. Ele permite aos consumidores armazenar e gerenciar uma quantidade ilimitada de arquivos online. Todos os arquivos são protegidos, utilizando mecanismos de criptografia e de controle de acesso. Os preços são de acordo com a quantidade de dados armazenados e o volume de dados transferidos para fora da nuvem.

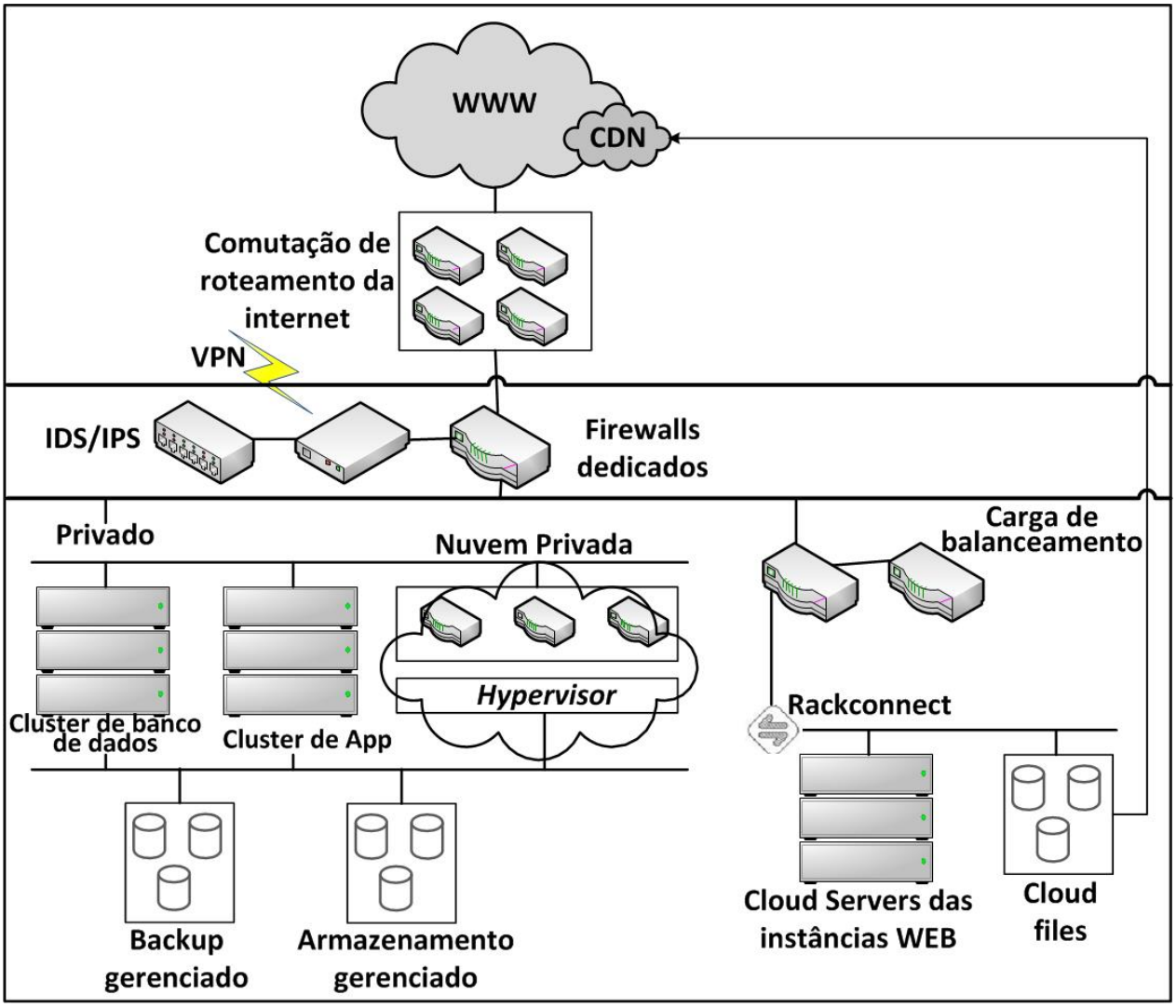
Diversas outras soluções são oferecidas pelo *Rackspace* aos seus clientes. De acordo com Gartner, que realiza a avaliação dos provedores de hospedagem, a *Rackspace* foi posicionada como líder nesse segmento de mercado [Gartner 2009]. Algumas arquiteturas são propostas pelo *Rackspace* para atender a demanda de seus clientes. No *Cloud Servers*, por exemplo, tem-se a arquitetura de nuvem básica, nuvem com carga balanceada, nuvem de alta disponibilidade, dedicada, híbrida e a híbrida de alta disponibilidade.

A arquitetura de Nuvem Híbrida de Alta Disponibilidade para hospedagem de sites será detalhada a seguir, a escolha dessa arquitetura se deu por ser uma arquitetura que provê alta disponibilidade aos usuários, um dos focos desta Dissertação de Mestrado. A Figura 2.7 ilustra a arquitetura completa da Nuvem Híbrida de Alta Disponibilidade da *Rackspace* [Rackspace 2013].

A configuração desta arquitetura apresenta [Rackspace 2013]:

- Alta disponibilidade.

Figura 2.7: Arquitetura de Nuvem Híbrida de Alta Disponibilidade *Rackspace*



Fonte: [Rackspace 2013]

- Segurança aprimorada.
- Armazenamento facilmente expansível em *Cloud Servers* e *Cloud Files*.
- Retaguarda dedicada e dispositivo de rede de alto desempenho.
- Configurações altamente complexas e personalizadas.

Os consumidores, quando contratam os serviços de algum provedor de Computação em Nuvem, assinam, no ato do fechamento do contrato, um SLA. Nesse contrato, são informadas todas as responsabilidades do provedor que presta os serviços e dos consumidores. A quebra desse contrato por alguma parte envolvida pode acarretar, por exemplo, pagamento de multas e até o cancelamento do contrato. Cada provedor de serviço apresentam o seu próprio SLA.

2.5 Acordo de Nível de Serviço (SLA)

Um dos maiores desafios da Computação em Nuvem é monitorar de forma eficiente o uso dos recursos computacionais, garantindo que aquilo que foi acordado entre o cliente e o provedor dos serviços seja cumprido. Para realizar esse controle entre os recursos computacionais adquiridos durante a assinatura do contrato e o seu efetivo é utilizado o SLA.

Em contratos de Computação em Nuvem, é utilizado o conceito de acordos em nível de serviço ou mundialmente conhecido como *Service Level Agreements (SLA)*. O SLA é utilizado para definir as responsabilidades tanto para os clientes quanto para os provedores. Nesse acordo, é definida a utilização dos recursos computacionais advindos de um provedor. De forma resumida, pode-se definir SLA como um documento assinado entre as duas ou mais entidades em que a descrição, a entrega e a cobrança dos serviços contratados são definidos formalmente [Marilly et al. 2002].

É possível identificar uma estrutura geral para um SLA, que envolve as partes envolvidas na negociação, os parâmetros de SLA, as métricas utilizadas para realizar os cálculos do SLA, os algoritmos utilizados para calcular essas métricas e quais as ações a serem tomadas, caso algum envolvido na negociação viole o acordo assinado [Schnjakin et al. 2010].

O SLA, na maioria das vezes, contém os seguintes itens sobre as informações dos serviços contratados: desempenho, gerenciamento de problemas, responsabilidade das partes, garantias, medidas emergenciais, planos alternativos, planos para soluções temporárias, relatórios de monitoramento, disponibilidade, prioridades de serviço, segurança, confidencialidade e cancelamento do contrato [Schnjakin et al. 2010].

Para alcançar o sucesso utilizando o SLA, é necessário utilizar de forma adequada parâmetros que possam identificar aspectos qualitativos do ambiente de Computação em Nuvem, para que possam ser utilizados durante o monitoramento para posteriormente, determinar o *Quality of service* ou Qualidade dos Serviços (QoS) do ambiente monitorado [Marilly et al. 2002].

Os atributos da dependabilidade devem ser descritos no SLA, porque devem ser acordados entre os provedores de serviços em nuvem e os usuários dos serviços, devendo ser negociáveis, para que os serviços possam atender aos requisitos dos clientes. A arquitetura proposta nesta Dissertação tem como objetivo auxiliar o monitoramento do SLA assinado entre o provedor e os clientes. Havendo violações no acordo, podem ser aplicadas multas ou indenizações. Os contratos devem especificar, de forma clara, um limiar que deve ser atendido para todos os requisitos acordados no contrato e as penalidades devem ser especificadas para cada tipo de falha apresentada pelo sistema.

Todas as definições e conceituação apresentadas até a aqui mostram a importância da Computação em Nuvem nas tecnologias atuais presentes no mercado de TI, mas a

Computação em Nuvem ainda apresenta muitos problemas e desconfianças por parte de muitos consumidores. À medida que estudos como esse são apresentados, essa desconfiança tende a diminuir e boa parte da população começa a confiar nos serviços que podem ser utilizados a partir do uso dessa tecnologia. A próxima seção apresenta os principais desafios encontrados pela Computação em Nuvem.

2.6 Desafios da Computação em Nuvem

No ambiente competitivo e rápido da TI, todas as mudanças apresentam medos e desconfianças. Com a Computação em Nuvem não foi diferente. Muitos profissionais e empresas de TI não tinham confiança nesse novo modelo de prestação de serviços, mas, com a evolução dos últimos anos nesse modelo, a confiança tem aumentado. A Computação em Nuvem ganha espaço como um modelo bem sucedido que oferece economia, flexibilidade e escalabilidade de recursos.

As organizações deixaram de ser preocupar com compra de super computadores e *software* e têm investido nesse novo modelo de negócio, cuja cobrança é realizada em relação aos recursos consumidos. Ainda assim, diversas empresas relatam que existe uma barreira em utilizar os serviços de Computação em Nuvem: eles se preocupam com a segurança e com a confidencialidade das informações. Ao utilizar a Computação em Nuvem, seus dados ficam armazenados em *data centers* compartilhados, e, na maioria das vezes, não se sabe a localização física desses *data centers*.

O problema citado acima é dos desafios encontrados na Computação em Nuvem. Como convencer os consumidores que seus dados estão seguros e que somente pessoas autorizadas a acessá-los tem acesso. Dez desafios e oportunidades foram levantados para melhorar a Computação em Nuvem [Armbrust et al. 2009]. A Tabela 2.3 mostra os 10 desafios e oportunidades da Computação em Nuvem. Estes desafios estão cada vez mais presentes nos dias atuais e possivelmente nos anos vindouros.

Fonte: [Armbrust et al. 2009]

Os dez desafios citados na Tabela 2.3 geram as novas oportunidades que devem ser trabalhadas para aumentar a utilização e a confiança nos serviços de Computação em Nuvem. Nas próximas seções são detalhadas de forma sucinta cada um dos dez desafios e as oportunidades que devem ser geradas com esses desafios.

2.6.1 Disponibilidade do serviço

Um dos principais desafios dos provedores de serviços em nuvem é a disponibilidade do serviço. Em diversas aplicações, sejam de modelo crítico ou não, a disponibilidade do serviço é importante para o funcionamento dos processos e negócios da corporação. Um monitoramento adequado da infraestrutura é importante para evitar que o sistema

Tabela 2.3: Os dez desafios e oportunidades para a Computação em Nuvem

	Desafios	**Oportunidades**
1	*Disponibilidade do serviço*	Utilizar vários provedores de nuvem para fornecer continuidade no negócio
2	*Dados em lock-in*	Padronização de APIs
3	*Confidencialidade e auditabilidade*	Implementação de algoritmos de criptografia, utilização de firewalls e VLANS
4	*Gargalos na transferência de arquivos*	Aumentar a largura de banda
5	*Imprevisibilidade no desempenho*	Monitoramento contínuo das máquinas virtuais, memória e disco
6	*Armazenamento escalável*	Para os usuários os recursos devem parecer infinitos
7	*Bugs em larga escala em sistemas distribuídos*	Debugar ambientes que se baseia em VMs distribuídas
8	*Escalonamento rápido*	Criar um escalonador automático que utilize aprendizado de máquina
9	*Lista de reputação*	Oferecer serviços de reputação
10	*Licença de software*	Pagamento pelo uso

fique indisponível, pois cada tempo de parada do sistema pode acarretar perdas financeiras enormes nos negócios. Uma saída para os sistemas de missão crítica é fechar contratos com dois provedores de serviços na nuvem e fazer a implementação da aplicação em ambos. Assim, os custos para utilização dos serviços aumentam, pois o custo com provedores cresce, em virtude da contratação desses dois provedores de serviços [Armbrust et al. 2009].

2.6.2 Dados em *lock-in*

A integração entre diversos provedores de serviços em nuvem ainda é algo a ser discutido. Cada provedor de serviço em nuvem utiliza suas próprias APIs, sem a possibilidade de interoperabilidade entre as infraestruturas. A proposta seria a padronização dos ambientes de aplicação e máquinas virtuais. Logo, a solução para esse desafio seria a padronização das APIs para que desenvolvedores de aplicativos PaaS possam integrar diferentes aplicações utilizando as várias infraestruturas de nuvem pública disponíveis no mercado [Armbrust et al. 2009].

2.6.3 Confidencialidade e auditabilidade

Um dos desafios mais complexos e difíceis de serem resolvidos na Computação em Nuvem é a confidencialidade e auditabilidade. Em nuvem pública, todas as informações dos usuários ficam alocadas em *data centers* de responsabilidade dos provedores de serviços em nuvem; essas informações podem vazar e causar prejuízos para uma organização. Então, diversas empresas ainda enxergam a nuvem com receio de que seus dados sejam vazados pelo provedor de serviço. Já em relação à auditabilidade, ainda não há consenso sobre como são realizadas as auditorias quando à infraestrutura, acessibilidade, segurança, entre outros tópicos importantes no ambiente de Computação em Nuvem. Uma oportunidade para solucionar os problemas com confidencialidade é a implementação de algoritmos de criptografia para os dados alocados na infraestrutura da nuvem. Também é valida a utilização de *firewalls* e *Virtual Local Area Network (VLANs)* [Armbrust et al. 2009].

2.6.4 Gargalos na transferência de arquivos

A maioria das aplicações que utilizam a infraestrutura de Computação Nuvem gera uma massa de dados que pode a chegar a níveis bastantes elevados. Logo, pode haver um gargalo no envio das informações necessárias para o funcionamento das aplicações. Esses dados podem levar um tempo elevado para trafegar pela *Internet*. Essa limitação técnica é um desafio apresentado pela Computação em Nuvem. Uma solução seria aumentar a largura de banda entre o provedor e os consumidores de serviços em nuvem [Armbrust et al. 2009].

2.6.5 Imprevisibilidade no desempenho

As máquinas virtuais compartilham os mesmos recursos computacionais, tais como, CPU, memória e rede e isso pode acarretar um *overhead* na utilização de um desses recursos e, consequentemente, pode ocasionar atraso na entrega da resposta das aplicações hospedadas nas nuvens. Para melhorar a previsão de desempenho, é necessário realizar um monitoramento contínuo da infraestrutura. Todos os recursos computacionais devem ser monitorados durante toda a vida útil do sistema, assim sendo, os administradores podem tomar decisões que colaborem com a melhor utilização dos recursos computacionais da infraestrutura do provedor de serviços de Computação em Nuvem [Armbrust et al. 2009].

2.6.6 Armazenamento escalável

A Computação em Nuvem é vista pelos seus consumidores como uma infraestrutura que pode oferecer recursos computacionais infinitos. Esses recursos podem ser alocados e desalocados à medida que os consumidores necessitam deles, sem a intervenção humana no sistema. O armazenamento escalável deve ser oferecido, de forma que os consumidores

realizem a solicitação de recursos e que suas requisições sejam atendidas, sem nenhuma falha. Caso uma demanda por mais espaço não seja atendida, certamente a empresa mudará para outro provedor que ofereça seus serviços sem essa restrição de recurso. O provedor de serviço em nuvem deve estar preparado para aumentar seu espaço de armazenamento, antes que seus clientes percebam esse problema. Os provedores de serviços de Computação em Nuvem devem estar preparados para atender à demanda de seus consumidores com alocação e desalocação de recursos à medida que são solicitadas pelos seus consumidores [Armbrust et al. 2009].

2.6.7 *Bugs* em larga escala em sistemas distribuídos

Os sistemas distribuídos, por apresentarem uma heterogeneidade, são complexos e é difícil analisar o seu comportamento. Além de desenvolver mecanismos de tolerância a falhas, exclusão mútua, integridade de dados, entre outros, é necessária a abstração de sintomas de comportamento da aplicação em ambientes de larga escala. O desenvolvimento de aplicações distribuídas, na maioria das vezes, é realizado em ambientes comuns, simulando o ambiente de produção. Ao hospedar essas aplicações em infraestruturas de Computação em Nuvem, podem-se apresentar situações não identificadas ou previstas durante o seu desenvolvimento. Muitos *bugs* são apresentados quando as aplicações são implantadas no ambiente distribuído. A proposta seria criar métodos para removê-los das aplicações antes de hospedar na infraestrutura de Computação em Nuvem, levando em consideração as características apresentadas nos ambientes distribuídos [Armbrust et al. 2009].

2.6.8 Escalonamento rápido

Os provedores de serviços de Computação em Nuvem devem apresentar um ambiente que seja capaz de realizar um escalonamento rápido dos seus recursos. A necessidade por mais recursos computacionais é proporcional à demanda apresentada pela aplicação e às requisições dos usuários que utilizam a aplicação. A desalocação de recursos funciona da mesma forma, tornando um custo operacional ao provedor, pois, quando o sistema está inativo, representa um recurso computacional disponível para processar, porém não há nenhum processamento. Logo, esse custo operacional é um problema ao provedor de serviços, porque é preciso utilizar os recursos computacionais com planejamento, deixando o processamento para os aplicativos ativos e desalocando recursos quando não há mais utilização. Uma proposta é criar um escalonador automático que disponibilize para a infraestrutura os recursos ociosos [Armbrust et al. 2009].

2.6.9 Lista de reputação

A lista de reputação é chamada de *Reputation Fate Sharing*, compartilhamento de reputação, que é uma lista de reputação por cliente. Um provedor de Computação em Nuvem oferece aos seus consumidores máquinas virtuais; existe o risco de esses consumidores utilizarem essas máquinas virtuais para assuntos ilegais, invasões, *spamming* e diversas outras coisas ilegais. Para evitar que esses problemas ocorram, os provedores precisam implantar mecanismos para evitar que aplicações dessa natureza cause algum dano ao seu ambiente. Também deve ser criado um documento com transferência de responsabilidades, em que os usuários dos serviços serão os responsáveis por qualquer processo por práticas ilícitas, caso não respeitem as políticas de segurança e privacidade do provedor de serviço [Armbrust et al. 2009].

2.6.10 Licença de *software*

A licença de *software* deve ser mais explorada pela indústria de desenvolvimento de programas. A maneira de realizar a cobrança na Computação em Nuvem é diferente, logo um novo modelo de negócio deve ser implantado pelas indústrias desenvolvedoras de *software*. Esse desafio já está bem adiantado, as grandes empresas desenvolvedoras de *software* já investem nesse novo modelo de negócios e, aos poucos, as médias e pequenas têm-se adequado a esse novo modelo de negócio. O método de cobrança realizado pelos provedores de serviços em nuvem é o pagamento pelo uso dos serviços.

2.7 Conclusão

Neste capítulo foram apresentados alguns conceitos fundamentais de Computação em Nuvem para a compreensão do trabalho proposto. Foram definidas as principais características existentes na Computação em Nuvem, os modelos de serviços e os modelos de implantação de nuvem definidos pelo NIST.

Na sequência, apresentou-se as arquiteturas de Computação em Nuvem existentes no mercado, detalhando as arquiteturas da Amazon e do Rackspace, que são os principais provedores de Computação em Nuvem. Tais provedores fornecem os serviços aos consumidores de acordo com o que foi assinado no SLA e, deste modo, a importância do SLA se mostra clara durante o fechamento dos contratos entre os provedores de serviços e os consumidores.

Além disso, este capítulo mostrou os principais conceitos de SLA. Ao final, também são apresentados dez desafios existentes na Computação em Nuvem que podem ser utilizados nas pesquisas sobre o tema.

Capítulo 3

Plataformas de gerenciamento e dependabilidade das nuvens

Neste capítulo, são descritas algumas plataformas de gerenciamento de ambientes de Computação em Nuvem, utilizadas para auxiliar na implantação e manutenção das infraestruturas de Computação em Nuvem. Em seguida, são detalhados os principais conceitos de dependabilidade, em relação à Computação em Nuvem.

3.1 Plataformas de Gerenciamento para Nuvem

Devido ao aumento do uso de tecnologias de virtualização, tem-se uma necessidade maior de administrar os recursos virtualizados utilizados pelos consumidores. Logo, surgiu a necessidade de utilizar um gerenciador de infraestrutura virtual. Esse gerenciador é responsável pelo ciclo de vida das máquinas virtuais.

A plataforma de gerenciamento de nuvem auxilia no gerenciamento dos componentes da infraestrutura virtualizada; é utilizada para configurar e operar infraestruturas de Computação em Nuvem. A plataforma de gerenciamento de nuvem oferece uma visão homogênea dos recursos virtuais, pois particularidades dos diversos *hypervisors* ficam transparentes aos usuários. O gerenciador tem como função principal disponibilizar aos usuários uma central de controle à qual, a qualquer momento, novos recursos podem ser adicionados, máquinas virtuais podem ser criadas, diversas redes virtuais podem ser configuradas e várias imagens são disponibilizadas [OpenStack 2013b].

Para a criação de uma nuvem no modelo IaaS, é imprescindível a utilização de um gerenciador de infraestrutura virtual. Existem várias plataformas que realizam o gerenciamento dos recursos da nuvem. Podemos citar algumas plataformas de gerenciamento de infraestrutura de nuvem com código aberto como: *Eucalyptus*, *OpenNebula* e o *OpenStack*.

3.1.1 *Eucalyptus*

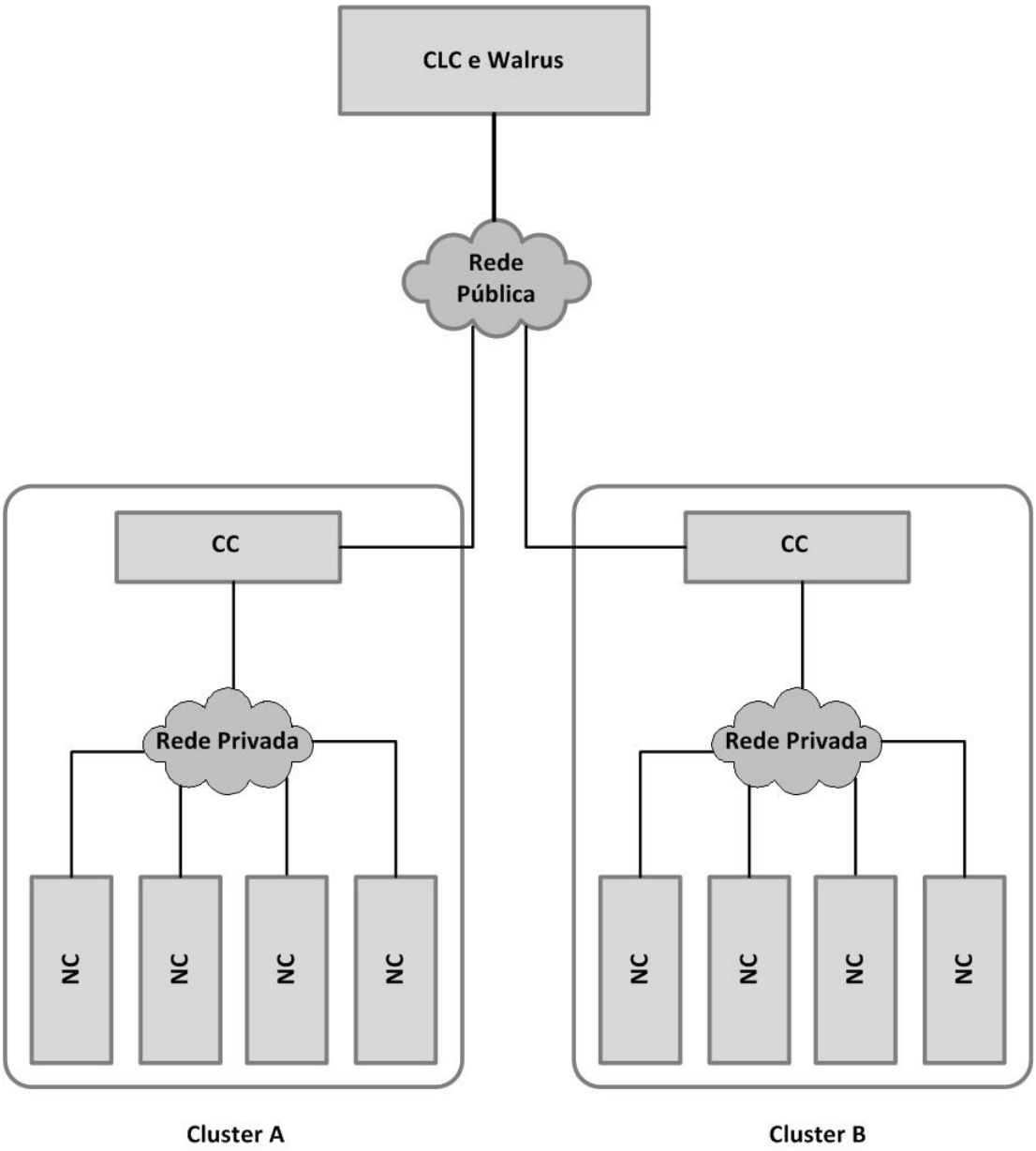
O *Eucalyptus (Elastic Utility Computing Architecture Linking Your Programs To Useful Systems)* foi um projeto desenvolvido pelo *Computer Science Department* da Universidade da Califórnia. A primeira versão foi lançada em 29 de maio de 2008 com suporte apenas a *EC2* e em dezembro de 2008 foi incluída a interface com o serviço *S3*. Em 2009, foi lançada a companhia *Eucalyptus Systems Inc.* para comercializar o *Eucalyptus Enterprise*. É uma infraestrutura de código aberto, como o *OpenStack*, que fornece uma interface compatível com o *Amazon EC2*, *Elastic Block Store (EBS)* e permite aos usuários criarem uma infraestrutura e experimentar a Computação em Nuvem. A Figura 3.1 mostra a arquitetura do *Eucalyptus* [Nurmi et al. 2009].

A arquitetura do *Eucalyptus* pode ser definida como um sistema simples, flexível e modular. O sistema suporta máquinas virtuais e utiliza o *hypervisor* Xen para realizar a virtualização. Sua arquitetura é composta por quatro componentes, como descrito a seguir [Nurmi et al. 2009]:

- ***Node Controller (NC)*:** Controla instâncias das máquinas virtuais nos nós. Pode-se alocar um NC em uma máquina física e diversos outros NCs rodando em máquinas virtuais. É responsabilidade do NC realizar a inspeção, execução e o término da respectiva instância.
- ***Cluster Controller (CC)*:** Ponte de comunicação entre NC e *Cloud Controller (CLC)*. É responsável por realizar o roteamento de pacotes entre as redes virtualizadas externa (pública) com a interna (privada). O CC é responsável por gerenciar todas as manipulações realizadas nos recursos físicos alocados nos NC.
- ***Storage Controller (Walrus)*:** Fornece acesso aos blocos de armazenamento em rede, como por exemplo *Amazon Elastic Block Storage (EBS)* e é capaz de interagir com sistemas de armazenamento, como o *Network File System (NFS)*, *Internet Small Computer System (iSCI)* e outros.
- ***Cloud Controller (CLC)*:** Controla a nuvem como um todo. É responsável por consultar o nó de controle para obter informações sobre os recursos físicos, tomar decisões de programação de alto nível e implementá-las por meio de requisições aos *Cluster Controller*.

O *Eucalyptus* permite aos usuários iniciar, controlar o acesso e gerenciar todas as máquinas virtuais utilizando uma emulação do Protocolo Simples de Acesso a Objetos (SOAP) do *Amazon EC2* e interfaces de consulta. Assim sendo, os usuários que utilizam *Eucalyptus*, trabalham com as mesmas ferramentas e interfaces que eles utilizariam com o *Amazon EC2*.

Figura 3.1: Arquitetura do *Eucalyptus*



Fonte: [Nurmi et al. 2009]

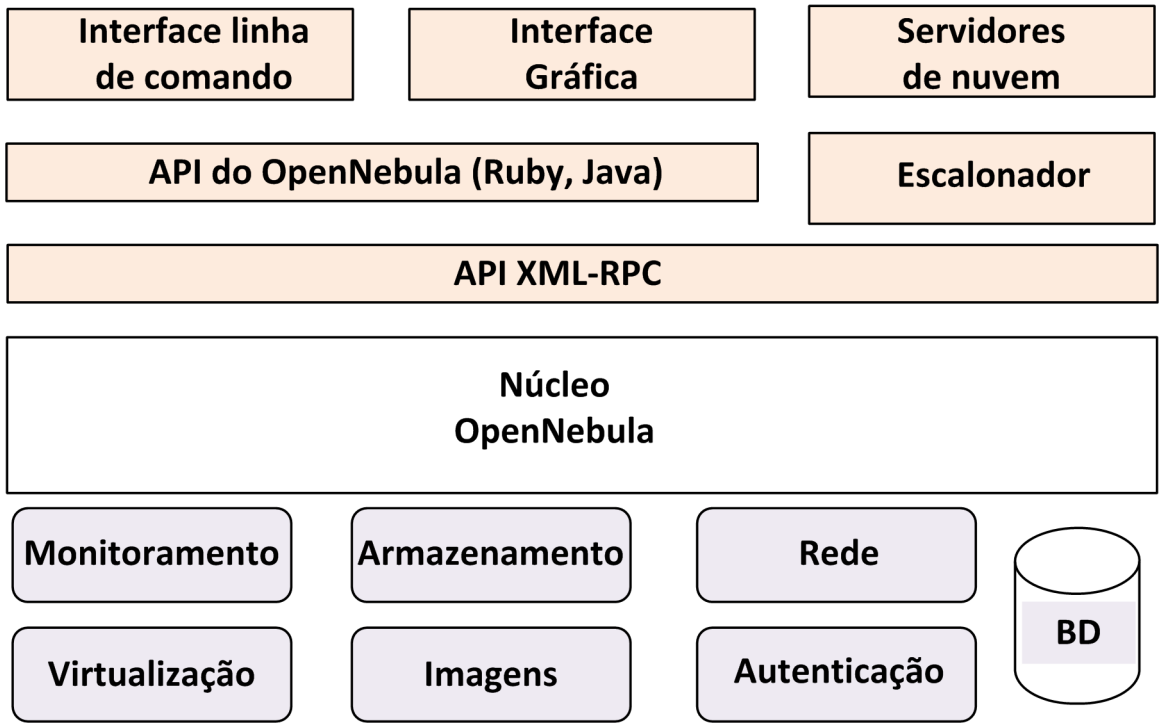
3.1.2 *OpenNebula*

O *OpenNebula* é uma plataforma de gerenciamento de infraestrutura virtual de código aberto, que oferece aos usuários o gerenciamento de todos os recursos computacionais oferecidos em cada servidor físico ou nó de forma transparente. O *OpenNebula* permite utilizar diferentes *hypervisors* como Xen, *Kernel-based Virtual Machine (KVM)* e VMWare [OpenNebula 2013].

O gerenciador de nuvem tem como responsabilidades realizar o controle e a criação de máquinas virtuais, bem como criar imagens. Ele também é responsável por realizar a migração entre máquinas virtuais, caso alguma falha ocorra em algum nó da infraestrutura. Com o *OpenNebula* o usuário pode separar o tráfego de informações entre as diferentes

aplicações hospedadas, por meio da criação de redes virtuais. Sendo assim, quando uma máquina virtual é criada, é possível configurar em que rede virtual essa nova máquina estará vinculada. A Figura 3.2 ilustra a arquitetura do *OpenNebula* [OpenNebula 2013].

Figura 3.2: Arquitetura do OpenNebula



Fonte: [OpenNebula 2013]

A arquitetura do *OpenNebula* é dividida em três camadas: escalonador, núcleo e *drivers*. Cada camada apresenta as suas características e tem suas funções.

A primeira camada é a do o escalonador, servidores e outras ferramentas. Essa camada apresenta diversas ferramentas fornecidas junto ao *OpenNebula*. As ferramentas são uma interface de linha de comando, uma interface gráfica e ferramentas próprias que podem ser criadas pelos usuários por meio das interfaces *eXtensive Markup Language (XML)*-Chamada de Procedimento Remoto (RPC) ou a API do *OpenNebula*. Também está presente nessa camada o escalonador que toma decisões sobre as alocações das máquinas virtuais. O escalonador tem acesso a todas as requisições que o *OpenNebula* recebe e de acordo processa essas requisições ele realiza as alocações atuais e futuras.

A segunda camada do *OpenNebula* é o núcleo, responsável por gerenciar todo o ciclo de vida das máquinas virtuais. Ele apresenta três diferentes áreas de gerenciamento: o gerenciamento de imagens e armazenamento que realiza a preparação de imagens de disco para as máquinas virtuais; o gerenciamento de rede que realiza o provimento de ambientes de rede para as máquinas virtuais; e o gerenciamento dos *hypervisors* para a criação e gerenciamento das máquinas virtuais. Essa camada realiza procedimentos de armazenamento, rede e virtualização por meio de *drivers* conectáveis. Logo, o *OpenNebula*

não está vinculado a qualquer ambiente específico, ele simplesmente fornece uma camada de gerenciamento uniforme, independentemente da infraestrutura utilizada.

A terceira e última camada é a de *drivers*. Essa camada fornece um conjunto de módulos conectáveis para interagir com *middlewares* específicos, como por exemplo, mecanismo de transferência de arquivos e *hypervisors* de virtualização. Esses módulos são chamados de D*rivers de Acesso Middleware*. A plataforma de gerenciamento de nuvem *OpenNebula* foi desenvolvida para ser modular, permitindo assim, uma integração com um maior número possível de *hypervisors* e diferentes arquiteturas de Computação em Nuvem [OpenNebula 2013].

3.1.3 *OpenStack*

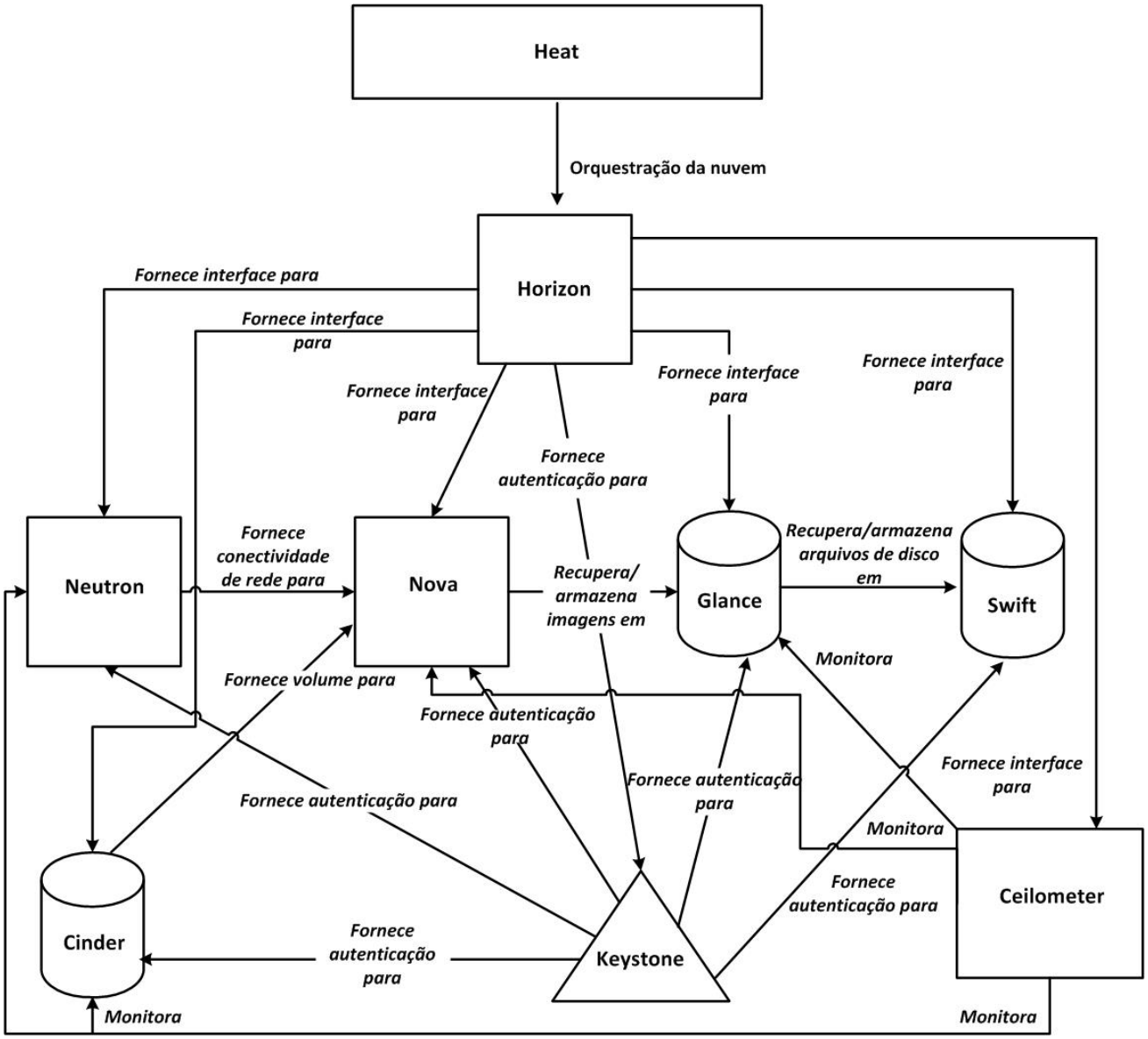
Outra plataforma de gerenciamento de nuvem existente no mercado é o *OpenStack*. Em Julho de 2010, a *Rackspace* e a *National Aeronautics and Space Administration* (NASA) se uniram e lançaram a plataforma conhecida como *OpenStack*, um projeto de código aberto que permite a criação de serviços de IaaS. Muitos citam o *OpenStack* como o Sistema Operacional da nuvem devido à sua capacidade de gerenciar recursos e componentes das infraestruturas virtualizadas. O projeto é administrado pela *OpenStack* Foundation, uma organização sem fins lucrativos.

Para a implementação da arquitetura proposta, foi utilizado esse gerenciador de nuvem. Ele é responsável pelo ciclo de vida das máquinas virtuais e pelo monitoramento dos recursos computacionais disponíveis. O gerenciador disponibiliza aos usuários um ponto de controle central, por meio dos quais novos recursos são adicionados, máquinas virtuais são criadas, redes virtuais são configuradas e novas imagens são disponibilizadas.

A arquitetura do *OpenStack* na versão *Havana* é constituída de nove componentes principais, a Figura 3.3 mostra seus componentes e os relacionamentos entre eles. A seguir a descrição de cada componente [OpenStack 2013c]:

- ***Dashboard(Horizon)***: disponibiliza uma interface web modular para todos os serviços do *OpenStack*, por meio da qual o cliente da nuvem gerencia seus recursos.
- ***Object Store (Swift)***: provê um serviço de armazenamento de objetos, por meio do qual os clientes podem armazenar ou buscar seus arquivos.
- ***Image Store (Glance)***: disponibiliza um catálogo e repositório de imagens de discos virtuais, que são utilizados pelas máquinas virtuais executadas nas nuvens.
- ***Compute (Nova)***: responsável por disponibilizar máquinas virtuais à medida que são demandadas pelos clientes.
- ***Identity (Keystone)***: provê autenticação e autorização para todos os serviços oferecidos pelo *OpenStack*.

Figura 3.3: Arquitetura do *OpenStack*



Fonte: [OpenStack 2013c]

- ***Networking (Neutron)***: fornece conectividade de rede como um serviço. Dá acesso a uma API para requerer e configurar redes virtuais. Esse componente permite que redes virtuais sejam configuradas entre os recursos de *compute* e *storage*.
- ***Block Storage (Cinder)***: fornece armazenamento em nível de bloco que pode ser montado como volume pelas instâncias (servidores virtuais).
- ***Ceilometer***: realiza o monitoramento de todos os componentes da infraestrutura.
- ***Heat***: Responsável pela orquestração de aplicações em nuvem utilizando o formato nativo ou o modelo compatível com o *Cloud Formation da AWS*.

3.1.4 Versões do *OpenStack*

Atualmente as versões do *OpenStack* são lançadas duas vezes ao ano. Na maioria dos casos é uma no mês de abril e outra no mês de outubro. Cada versão do *OpenStack* recebe um nome que tem a primeira letra baseada na ordem do alfabeto, como é ilustrado na Tabela 3.1.

Tabela 3.1: Versões do *OpenStack*

Versão	**Ano**	**Informações**
Austin	2010	Primeira versão do *OpenStack* com a junção dos projetos *Nova* e *Swift*
Bexar	2011	Adicionou-se o projeto *Glance*
Cactus	2011	Aprimorou os projetos
Diablo	2011	Foi lançado seis meses após o *Cactus*
Essex	2012	Incorporou dois novos projetos o *Keystone* e o *Horizon*
Folson	2012	Lançou dois novos projetos o *Cinder* e o *Quantum*
Grizzly	2013	Trouxe mais robustez ao *Cinder* e avanços ao *Quantum*
Havana	2013	A versão atual, que se utilizou neste trabalho. Nessa versão foram adicionados os projetos *Ceilometer* e *Heat*

3.1.5 *Dashboard (Horizon)*

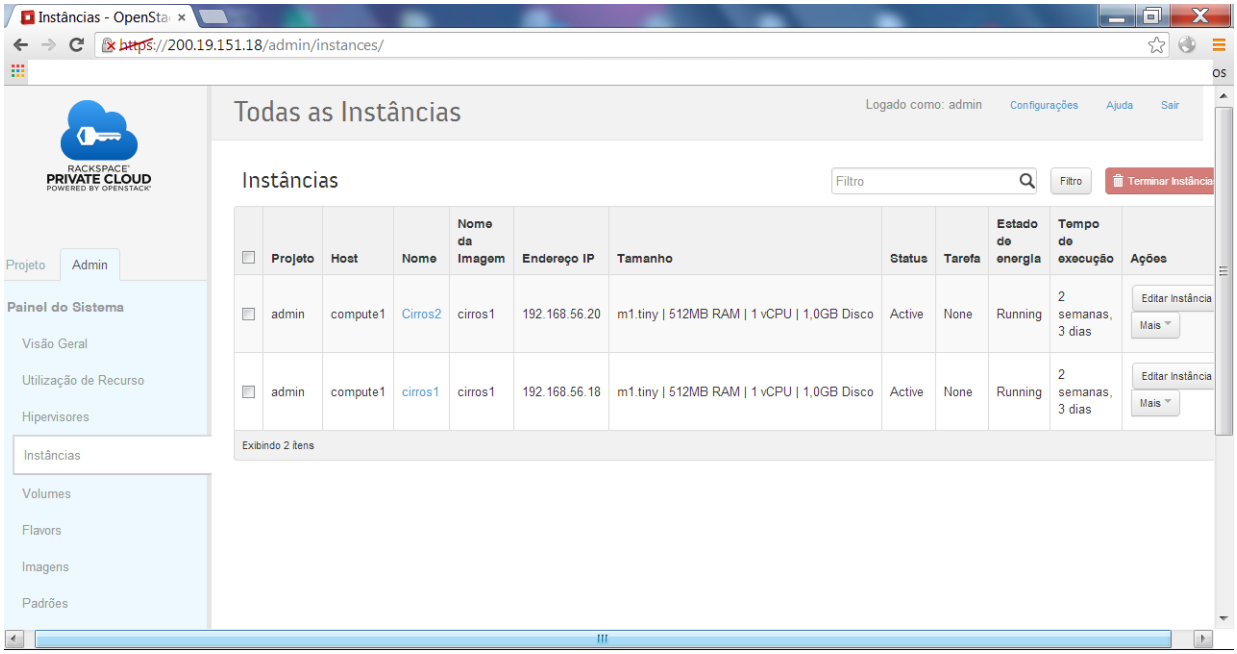
O *Dashboard*, também conhecido como codinome *Horizon*, é um componente que disponibiliza um painel de administração de serviços do *OpenStack*, por meio de uma interface Web. O *Horizon* é utilizado tanto por usuários finais, que gerenciará apenas volumes e instâncias, como também pelos administradores do ambiente. A comunicação é feita por meio de chamadas específicas de cada API, admitindo várias ações entre os usuários e os respectivos componentes. O acesso é via o Protocolo de Transferência de Hipertexto (HTTP) [OpenStack 2013c]. As principais funcionalidades desse componente são:

- **Gerenciamento de Instâncias:** permite a execução ou término de instâncias, o acesso de máquinas por Virtual Network Computing (VNC), anexar volumes, o uso de variáveis e outros.
- **Gestão de segurança:** possibilita a criação de grupos de segurança, par de chaves para acesso, atribuição de IPs flutuantes e outros.
- **Gerenciamento dos perfis de instâncias:** possibilita criar e apagar modelos de máquinas virtuais.
- **Gerenciamento de imagens:** permite editar ou apagar imagens de máquinas virtuais.

- **Gerenciamento de usuários:** permite a criação de novos usuários e define as atribuições ao inquilino.
- **Gerenciamento de volumes:** suporte para a criação de volumes.
- **Manipulação do *Object Store*:** possibilidade de criar e apagar *containers* e objetos.

A Figura 3.4 ilustra uma tela do *Dashboard*. Nessa tela são apresentadas as instâncias que estão configuradas na infraestrutura. Para acessar o *Dashboard* é necessário um usuário e senha.

Figura 3.4: Tela do *Dashboard*



Fonte: *OpenStack*

O *Dashboard* possui duas abas, a aba Projeto e Administrador. A aba Projeto é vista e acessada por todos os usuários cadastrados no *OpenStack*. Nela, as instâncias, volumes e outros recursos só podem ser acessados por usuários que pertençam ao mesmo projeto em que esses recursos foram criados. A aba Administrador só pode ser acessada por usuários que possuam a função de administrador. Essa aba possui diversas funções de administração na nuvem e tem acesso aos recursos criados em qualquer projeto. O acesso deve ser restrito, pois o uso indevido pode provocar falhas nos projetos.

As principais funções disponíveis na aba Projeto são:

- Administrar instâncias.
- Administrar volumes.
- Administrar *snapshots*.

- Administrar grupos de segurança.
- Administrar *storage* de objetos.
- Administrar par de chaves.
- Administrar IPs flutuantes.

Todas as funções citadas somente são criadas por usuários do próprio projeto.
As principais funções disponíveis na aba Administrador são:

- Administrar instâncias.
- Administrar volumes.
- Administrar *flavors*.
- Administrar imagens.
- Administrar projetos.
- Administrar usuários.
- Administrar funções.
- Administrar cotas.

3.1.6 *Object Store (Swift)*

O *Object Store*, também conhecido pelo codinome *Swift*, é um sistema totalmente distribuído, redundante e com alta disponibilidade para armazenamento de objetos. Ele é um armazenador global e redundante, sendo capaz de armazenar bilhões de objetos distribuídos entre os nós. O *Swift* não disponibiliza um volume para ser montado como um sistema de arquivos, e sim, um sistema de armazenamento para dados permanentes. Ele pode ser utilizado para o armazenamento de imagens de máquinas virtuais, armazenador de fotos, e-mails, *backups* e outros [OpenStack 2013c]. Seus componentes são:

- ***Swift Proxy Server:*** o *Proxy Server* é a ponte utilizada pelos usuários finais para se comunicarem com os componentes do *Swift*. Essas requisições podem ser feitas por API ou por meio do protocolo HTTP. A principal função do *Proxy Server* é verificar a localização dos nós responsáveis por armazenar os *objects*, *containers* ou *accounts* e distribuir as requisições para seu respectivo nó.
- ***Swift Object Server:*** é o componente que gerencia objetos (arquivos binários) nos servidores que possuem discos de armazenamento. Suas funções são as de salvar, recuperar e apagar os objetos no disco local. O *Object Server* suporta diversos tipos de sistemas de arquivos, mas é recomendado que se utilize o XFS. Vários nós do *Object Server* possuem cópias em outros nós, nativamente o *Object Server* replica um objeto em três nós diferentes.

- ***Swift Container Server:*** mapeia os objetos de cada *Container Server* em uma lista; essas listas são gravadas como arquivos de banco de dados *MySQL*. Tem a função de fornecer estatísticas de utilização, número total de objetos, quantidade de utilização e espaço de armazenamento total. Ele realiza a cópia dos mapeamentos em diferentes nós.
- ***Swift Account Server:*** mapeia os *containers* de respectivas contas de usuários. Este componente também realização a duplicação dos dados em nós distintos.

3.1.7 *Image Store (Glance)*

O *Glance*, como é conhecido o *Image Store*, tem a função de gerenciar as imagens de máquinas virtuais usadas pelo *Nova*. Além disso, o projeto de imagens do *OpenStack* é capaz de gerar *snapshots*, imagens instantâneas de máquinas virtuais, gerando novas imagens como *backup* da instância em execução.

Os componentes que fazem parte do *Glance* são o *glance-api*[1] e o *glance-registry*[2]. O *Glance* mantém um catálogo e o repositório das imagens, disponibilizando para que o *Nova* inicie as instâncias das máquinas virtuais [OpenStack 2013c]. Várias instâncias rodam a partir de uma cópia da imagem, logo, as alterações efetuadas nas instâncias não afetam a imagem original.

3.1.8 *Compute (Nova)*

O *Compute*, também conhecido como codinome *Nova*, é um dos principais componentes da estrutura do *OpenStack*. Ele está presente desde da primeira versão e é responsável pelo gerenciamento da infraestrutura computacional da nuvem, fornecendo e controlando as máquinas virtuais sob demanda. Utilizando o *Nova*, é possível executar instâncias de máquinas virtuais, gerenciar a rede, alocar recursos e fazer autorizações de acesso à nuvem [OpenStack 2013c]. O *Nova* tem as seguintes responsabilidades:

- Gerenciamento do ciclo de vida das instâncias.
- Gerenciamento de todos os recursos computacionais.
- Rede e autorização.
- API REST (Integração administrativa e pública).
- Comunicação assíncrona.
- Integração com *hypervisors*.

[1]Interface que interage com as requisições dos usuários e com os dispositivos de armazenamento
[2]Serviço que efetua o mapeamento das imagens com os respectivos dispositivos de armazenamento

3.1.9 *Identity (Keystone)*

O *Keystone* realiza o controle de identificação e acesso para todos os componentes do *OpenStack*. Um usuário realiza a sua autenticação utilizando o *Keystone*, garantindo assim, que uma solicitação vem realmente de quem ele diz ser e também a sua autorização. Ele assegura que o usuário terá acesso somente aos serviços que está de fato solicitando. Possui uma própria API baseada no REST API, denominada *Indentify API* [OpenStack 2013c]. O *Keystone* tem como funções principais:

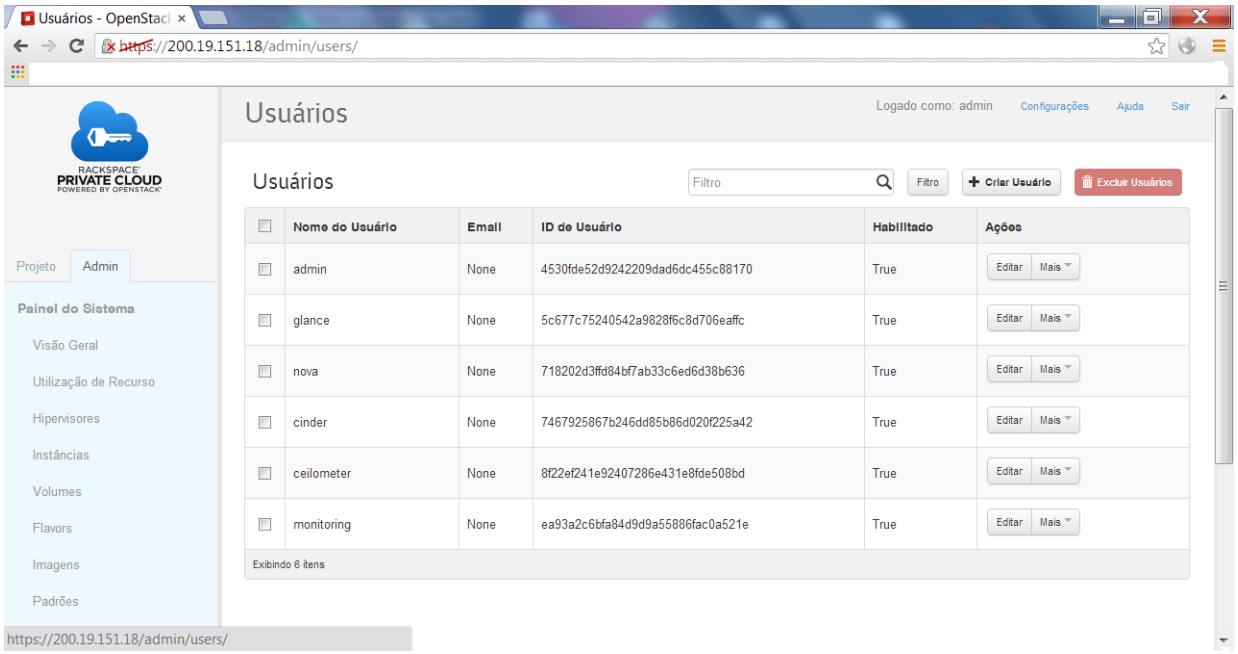
- O gerenciamento de usuários mantém um cadastro de usuários para autenticar e autorizar a realizarem ações.
- Tem o serviço de catálogo, possui uma lista de serviços associados aos componentes do *OpenStack* e que estão à disposição dos usuários.
- Políticas de serviços realizam a gerência dos serviços para o acesso específico de usuários ou grupos.

Uma autenticação de um usuário pode ser realizada de duas maneiras no *Keystone*, baseado na utilização de usuário com uma senha ou mediante a um *token*[3]. Alguns componentes do *Keystone* são:

- ***user:*** o usuário é a representação digital de um cliente de um determinado serviço do *OpenStack*. Ele possui informações associadas como senha ou e-mail. A Figura 3.5 mostra os usuários criados. É possível criar e vincular um usuário a um projeto ou excluir usuários do projeto.
- ***endpoints:*** é um endereço de rede (URL) utilizado para acessar um determinado serviço do *OpenStack* (*Nova*, *Swift*, *Glance*, outros).
- ***regions:*** toda localização física dedicada a um *data center*, como um servidor ou centro de dados dedicado.
- ***services:*** cada componente do *OpenStack* (*Nova*, *Swift*, *Glance*, outros) que está conectado, ou sendo administrado pelo *Keystone*. Cada componente do *OpenStack* fornece um ou mais *endpoints* em que os usuários podem acessar recursos e executar as operações.
- ***role* ou papéis:** uma regra ou conjunto de regras que podem ser associadas a diferentes usuários. De uma forma simples, é a função que um usuário assume e contém as permissões e as operações que o usuário pode executar em um determinado projeto. A Figura 3.6 mostra os papéis definidos na infraestrutura.

[3]Chave eletrônica utilizada para autenticação

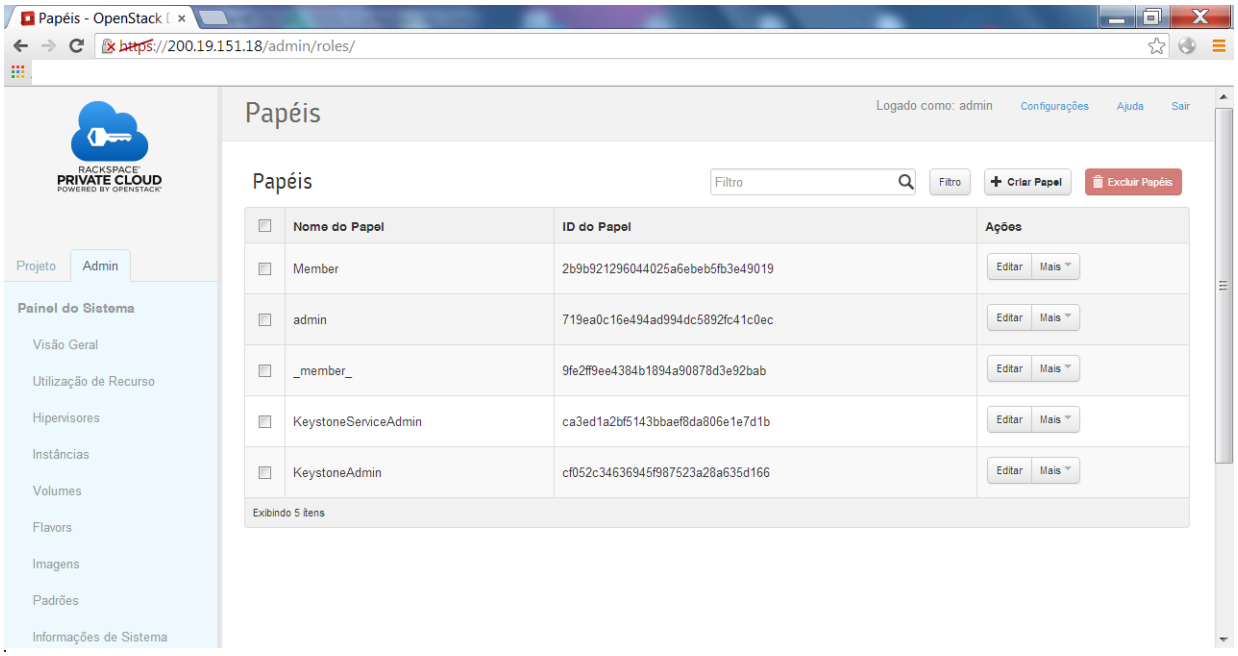
Figura 3.5: Tela do *Dashboard* para visualizar os usuários



Fonte: *OpenStack*

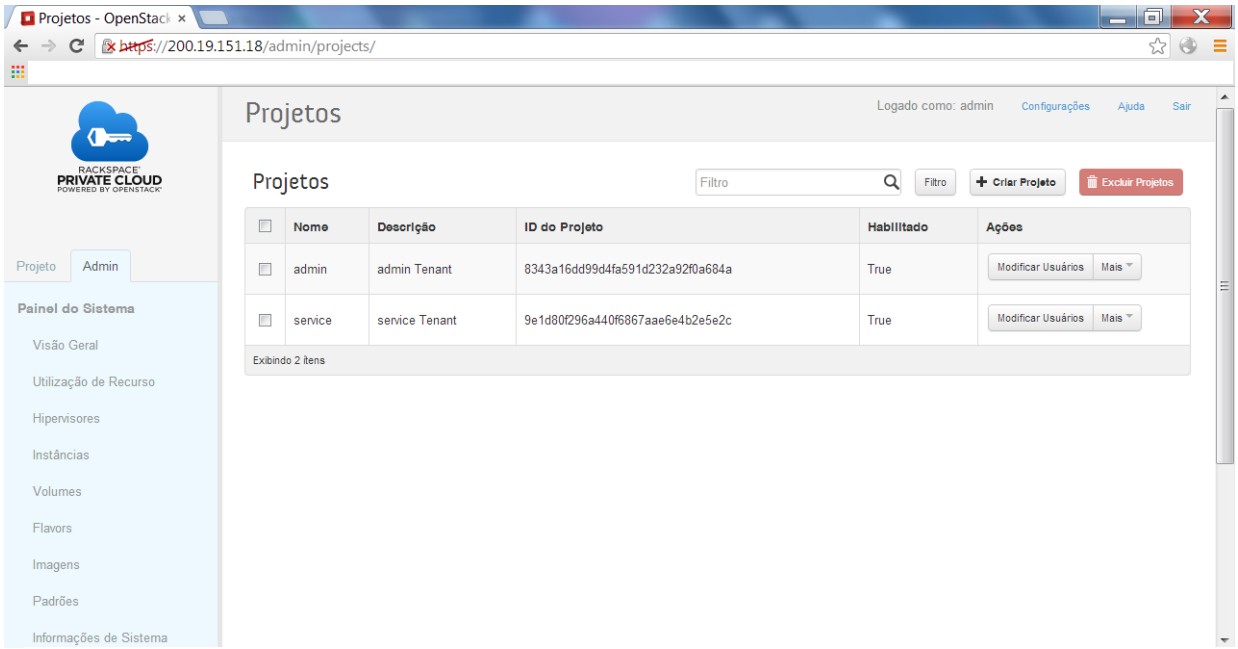
- ***tenant:*** é um projeto, representa uma conta, departamento ou empresa. Quando um usuário faz requisições a um serviço do *OpenStack*, é necessário especificar o projeto. Um exemplo é se um usuário solicitar a lista de instâncias ativas, vai receber a lista de instâncias ativas do projeto que especificou. A Figura 3.7 ilustra os projetos no *Dashboard*.

Figura 3.6: Tela do *Dashboard* para visualizar os papéis



Fonte: *OpenStack*

Figura 3.7: Tela do *Dashboard* para visualizar os projetos



Fonte: *OpenStack*

Os usuários, com as suas devidas permissões, podem criar, excluir e editar os projetos existentes. Através do *Dashboard* é possível visualizar e administrar os recursos do *Keystone*.

3.1.10 *Networking (Neutron)*

O *Networking* tem como codinome o *Neutron*. É o componente do *OpenStack* responsável pelo gerenciamento da rede na nuvem. Ele provê "Rede Como Um Serviço"para os demais componentes do *OpenStack*. Ele tem como principal objetivo utilizar, de maneira escalável, os recursos de rede disponível em um *cluster*. Para isso, criam-se redes virtuais que atendam a necessidade do usuário.

Diversas topologias de redes podem ser desenhadas utilizando o *Neutron*, sub-redes e portas. Existem também *plugins* para os controladores de interface de rede implementadas nos *software* de virtualização gerenciados pelo *Nova*. Ele também é responsável pelo endereçamento IP, de prover agentes do Protocolo de Configuração Dinâmica de Host (DHCP) e Tradução de Endereços de Rede (NAT) ou IPs estáticos [OpenStack 2013c].

3.1.11 *Block Storage (Cinder)*

O *Block Storage*, tem o codinome de *Cinder*. Ele faz o gerenciamento de blocos de disco na nuvem. Um bloco de disco criado pelo *Cinder* é um disco sem partição e formatação, cabe a instância que o utiliza formatá-lo e particioná-lo na primeira vez em que for utilizar [OpenStack 2013c]. As principais funções do *Cinder* são:

- Criação e remoção de volumes de disco.
- Criação e remoção de "estado"de disco (*snapshot*).
- Criação de um volume a partir de um *snapshot*.
- Recuperação de metadados do volume.

Existem dois tipos de armazenamento o *ephemeral* e o de volume. O armazenamento *ephemeral* é referente a uma instância, em que o sistema de arquivos de uma imagem de máquina virtual é montada. Quando uma instância é terminada, o armazenamento é liberado. O armazenamento de volume é o persistente e independente de instâncias. Podem ser de qualquer tamanho, bastando somente escolher a formatação da partição. O volume persistente é criado pelo usuário e seu tamanho pode variar até a cota definida para o projeto. Pode ser anexado a uma instância como se fosse um disco rígido adicional, podendo somente estar anexado a uma instância por vez, mas pode ser desconectado de uma instância e conectado a outra.

3.1.12 *Heat*

O novo projeto *Heat*, lançado na versão Havana, implementa orquestração, fluxos de configuração de infraestrutura que podem ser customizados e automatizados para perfis de diferentes aplicações que rodam em ambientes de Computação em Nuvem. Esse componente tem como função realizar a orquestração de aplicações em nuvem, utilizando o formato nativo ou o modelo compatível com o *Cloud Formation da AWS*. Foram projetas APIs próprias e compatíveis com as utilizadas pela *AWS*.

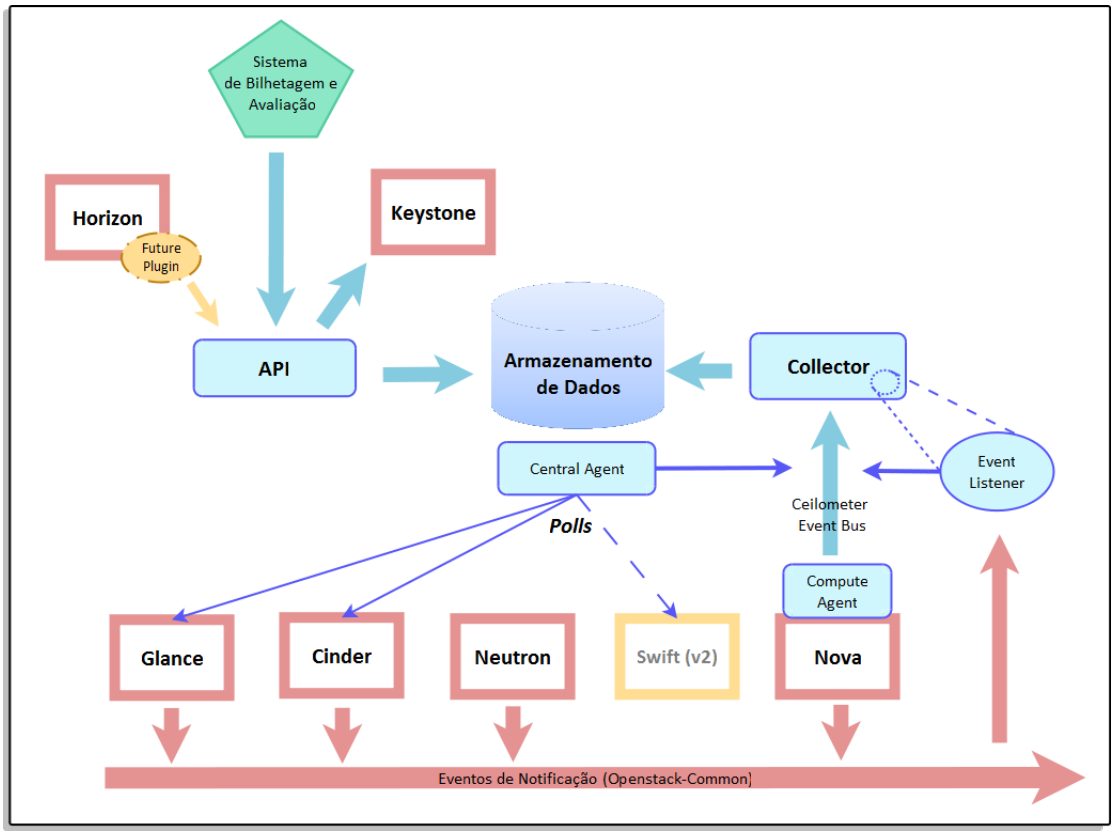
3.1.13 *Ceilometer*

Esse projeto foi iniciado em 2012 com o objetivo de oferecer uma infraestrutura para coletar informações e métricas a respeito dos componentes do *OpenStack* [OpenStack 2013d]. As métricas como CPU, memória, rede, disco e energia são coletadas por meio do *Ceilometer*. Essas informações coletadas são importantes para definir o que afeta a dependabilidade do sistema, que é um dos focos desta Dissertação de Mestrado. A arquitetura lógica do *Ceilometer* é resumida pela Figura 3.8.

A arquitetura do *Ceilometer* é composta de cinco componentes básicos [OpenStack 2012]:

1. ***Compute agent:*** Em cada nó um agente computacional é executado para coletar as estatísticas de utilização dos recursos.
2. ***Central agent:*** Um agente central é executado no servidor de gerenciamento para coletar estatísticas de utilização de recursos e enviar os resultados para o coletor.

Figura 3.8: Arquitetura do *Ceilometer*



Fonte: [OpenStack 2013d]

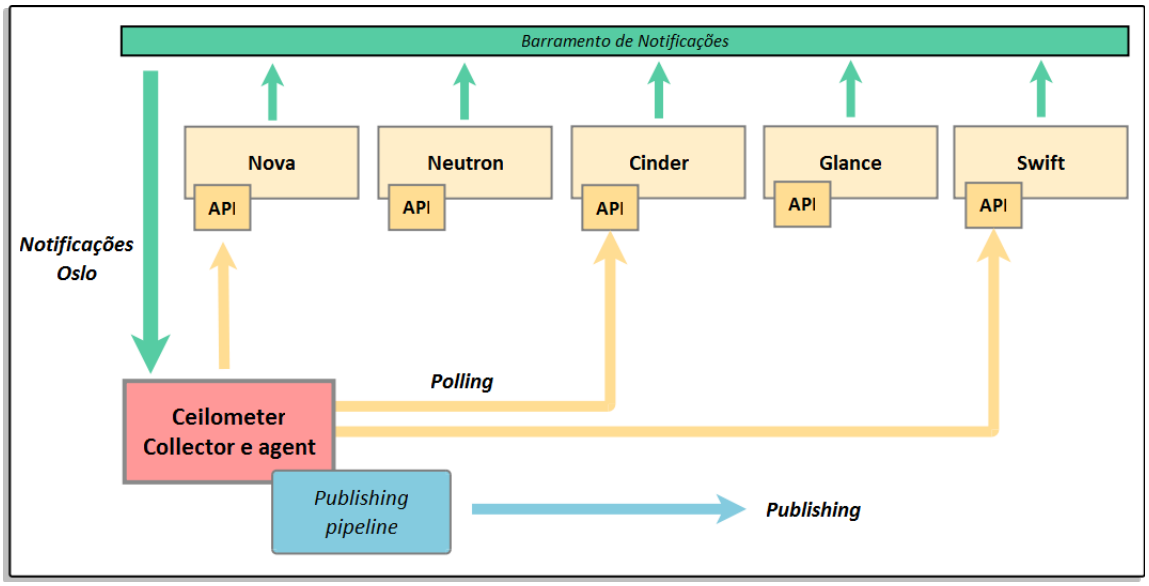
3. ***Collector:*** Um coletor é executado em um ou mais servidores de gerenciamento central para monitorar as filas de mensagens. As mensagens de notificação são processadas e transformadas em mensagens de medição e são enviadas de volta para barramento de mensagens. As mensagens de medição são gravadas no banco de dados sem sofrer nenhuma modificação.

4. ***Data Store:*** É uma base de dados capaz de lidar com a escrita e leitura simultânea de uma ou mais instâncias do *Collector*.

5. ***Server API:*** Um servidor de API funciona em um ou mais servidores de gerenciamento central para fornecer acesso aos dados armazenados na base de dados.

O *OpenStack* é uma coleção de componentes que cooperam entre si, para fornecer uma infraestrutura de serviços em nuvem. Cada componente tem uma função específica e utiliza um barramento de mensagens para comunicar entre si. Logo, todos os componentes geram mensagens de notificações quando um evento ocorre. Estas mensagens são uma das fontes de dados para o *Ceilometer*. Na versão *Havana*, existem três formas de coletar os dados utilizando o *Ceilometer*. A Figura 3.9 representa como os coletores e os agentes colhem os dados de múltiplas fontes. Os três métodos independentes de realizar a coleta dos dados são [OpenStack 2013d]:

Figura 3.9: Coleta de dados pelo *Collector* e *Agent*



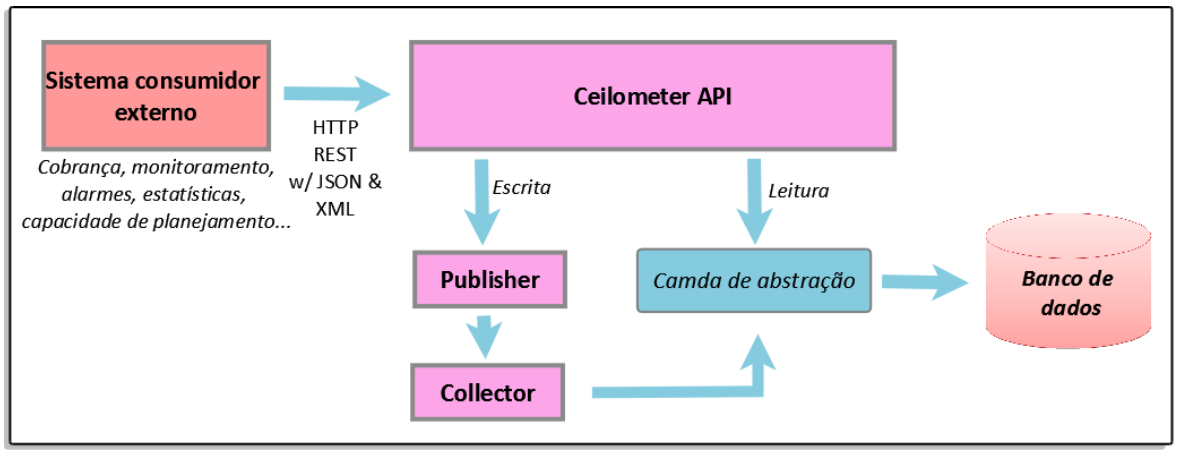
Fonte: [OpenStack 2013d]

1. *Bus listener agent:* o barramento de notificações coleta as informações dos eventos gerados e os transforma em amostras no *Ceilometer*. Esse é o método preferido para coletar os dados.
2. *Push agents:* é a única solução para coletar dados de projetos que não utilizam acesso remoto para obter as suas informações. Esse método não é o preferido, já que torna a implantação um pouco mais complexa, pois é necessário adicionar um componente em cada nó que tem de ser monitorado.
3. *Polling agents:* é o método menos preferido, ele utiliza alguma API ou outra ferramenta para coletar informações em um intervalo regular.

O barramento de notificações é o responsável de coletar as informações em todos os componentes do *OpenStack* e entregar ao *Ceilometer*. O Agente envia os resultados dos dados coletados para o Coletor. Já o *Collector* é um serviço de *software* que executa as notificações sobre a infraestrutura dos componentes do *OpenStack* e das amostras do Agente *Ceilometer*, e registra os resultados no banco de dados. O *Pipeline* de Publicação são as modificações que as métricas passam para ser publicadas externamente. Por exemplo, a frequência de publicação; assim, ao realizar a implementação do *Ceilometer* métodos diferentes de frequência e transporte podem ser implementados.

A Figura 3.10 apresenta a forma de acessar os dados armazenados pelo *Ceilometer*. O *Ceilometer* após coletar os dados dos componentes da infraestrutura, ele armazena essas informações coletadas.

Figura 3.10: Acesso aos dados armazenados pelo *Ceilometer*



Fonte: [OpenStack 2013d]

Após os dados serem coletados, são armazenados geralmente em um banco de dados ou em arquivos simples. Vários tipos de banco de dados podem ser utilizados. Acessam-se esses dados coletados, utilizando a API REST.

3.1.14 Método de comunicação

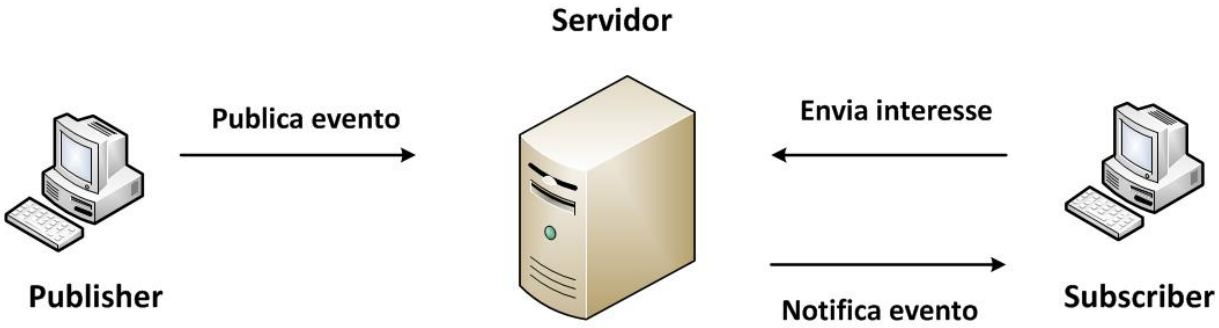
O método de comunicação utilizado nessa arquitetura é o *Publish-Subscribe*. Esse método de comunicação é baseado na troca assíncrona de mensagens, denominadas como eventos. O método consiste de um conjunto de clientes que publicam esses eventos (produtores), que são encaminhados para os clientes que registram interesse em recebê-los (consumidores). Ou seja, os usuários publicam a ocorrência de eventos e declaram seus interesses para que sejam notificados das ocorrências dos seus eventos de interesse [Eugster et al. 2003].

Os que publicam os eventos são chamados de *Publishers* e os consumidores são chamados de *Subscribers*, esses declaram seus interesses. A Figura 3.11 ilustra o funcionamento do método de comunicação *Publish-Subscribe* [Eugster et al. 2003].

Analisando a Figura 3.11, por exemplo, tem-se dois dispositivos um *Subscriber* e outro *Publisher*, o *Subscriber* é chamado de dispositivo A e o *Publisher* de dispositivo B. O dispositivo A quer receber uma notificação de algum evento que acontece no dispositivo B. Então, o dispositivo A envia uma notificação de interesse e espera que em um determinado momento o dispositivo publique a ocorrência desse evento. Assim que o servidor receber uma publicação do dispositivo B, o dispositivo A é notificado.

O *Ceilometer* permite a criação de *plugins* para coletar eventos que tenham maior interesse e ainda não estejam disponíveis para utilização na comunidade. Os desenvolve-

Figura 3.11: Funcionamento do método *Publisher-Subscriber*



Fonte: Adaptado [Eugster et al. 2003]

dores explicam que mesmo com uma lista grande de métricas que podem ser coletadas pelo *Ceilometer*, é impossível prever como os provedores de nuvem vão medir a utilização da infraestrutura pelos seus clientes. Então, o *Ceilometer* oferece essa flexibilidade de ajuste para atender a cada instalação. Três tipos de métricas são definidas pelo *Ceilometer* [OpenStack 2012], conforme ilustra a Tabela 3.2.

Tabela 3.2: Tipos de métricas

Tipo	**Definição**
Cumulative	Crescente ao longo do tempo (horas)
Gauge	Itens discretos (IPS flutuantes, upload de imagens) e valores flutuantes (disco E/S)
Delta	Sofre variação ao longo do tempo (banda larga)

Na Tabela 3.3 são descritas as unidades utilizadas pelo *Ceilometer*. Em algumas métricas, as unidades de medidas nunca devem ser alteradas. De maneira geral, quando uma métrica não representar volume, a descrição da unidade deve ser descrever o que é medido. Novas unidades de medidas podem ser criadas, desde que não haja similaridade com as existentes e devem ser inseridas essas novas métricas no código fonte.

Tabela 3.3: Unidades

Dimensão	**Unidade**	**Abreviações**
Não definido	N/A	
Volume	byte	B
Tempo	segundos	s

A seguir têm-se as métricas de todos os componentes do *OpenStack* que serão implementados na arquitetura proposta:

- **Métricas do *Swift***

  A Tabela 3.4 mostras as métricas disponíveis no *Ceilometer* para o componente *Swift*.

Tabela 3.4: Métricas do *Swift*

Nome	Tipo	Unidade	Recurso	Nota
storage.objects	*Gauge*	*objetos*	*imagem ID*	Número de objetos
storage.objects.size	*Gauge*	*B*	*imagem ID*	Tamanho total dos objetos armazenados
storage.objects.contai.	*Gauge*	*containers*	*imagem ID*	Número de containers
storage.objects.incoming	*Delta*	*B*	*imagem ID*	Número de bytes de entrada
storage.objects.outgoing	*Delta*	*B*	*imagem ID*	Número de bytes de saída
storage.api.request	*Delta*	*requisições*	*imagem ID*	Número previsto de requisições *Swift*

- **Métricas do *Nova***

  As métricas que podem ser coletadas no componente *Nova* são apresentadas na Tabela 3.5.

Tabela 3.5: Métricas do *Nova*

Nome	Tipo	Unidade	Recurso	Nota
instance	*Gauge*	*instância*	*inst ID*	Duração da instância
instance:<type>	*Gauge*	*instância*	*inst ID*	Duração da instância <tipo>
memory	*Gauge*	*MB*	*inst ID*	Volume da RAM em MB
cpu	*Cumulative*	*ns*	*inst ID*	Tempo de uso da CPU
cpu_util	*Gauge*	*%*	*inst ID*	Utilização média da CPU
vcpus	*Gauge*	*vcpu*	*inst ID*	Número de VCPUs
disk.read.requests	*Cumulative*	*requisições*	*inst ID*	Número de requisições de leitura
disk.read.requests	*Cumulative*	*requisições*	*inst ID*	Número de requisições de escrita
disk.read.bytes	*Cumulative*	*B*	*inst ID*	Volume de leitura em B
disk.write.bytes	*Cumulative*	*B*	*inst ID*	Volume de escrita em B
disk.root.size	*Gauge*	*GB*	*inst ID*	Tamanho do disco de root em GB
disk.ephemeral.size	*Gauge*	*GB*	*inst ID*	Tamanho do disco ephemeral em GB
network.incoming.bytes	*Cumulative*	*B*	*iface ID*	Número de bytes de entrada na rede de uma interface MV
network.outgoing.bytes	*Cumulative*	*B*	*iface ID*	Número de bytes de saída da rede para uma interface MV
network.incoming.packets	*Cumulative*	*pacotes*	*iface ID*	Número de pacotes de entrada para uma interface MV
network.outgoing.packets	*Cumulative*	*pacotes*	*iface ID*	Número de pacotes de saída para uma interface MV

- **Métricas do *Cinder***

  A Tabela 3.6 mostras as métricas disponíveis no *Ceilometer* para o componente *Cinder*. Sempre que um volume for medido, unidades e seu respectivo símbolo devem ser utilizados. A unidade deve ser expressa em *bits(b)* ou *bytes (B)*.

Tabela 3.6: Métricas do *Cinder*

Nome	Tipo	Unidade	Recurso	Nota
volume	*Gauge*	*volume*	*vol ID*	Duração do volume
volume.size	*Gauge*	*GB*	*vol ID*	Tamanho do volume

- **Métricas do *Neutron***

  As métricas que podem ser coletadas no componente *Neutron* são apresentadas na Tabela 3.7.

Tabela 3.7: Métricas do *Neutron*

Nome	Tipo	Unidade	Recurso	Nota
network	*Gauge*	*rede*	*netw ID*	Duração da rede
network.create	*Delta*	*rede*	*netw ID*	Criação de requisições para a rede
network.update	*Delta*	*rede*	*inst ID*	Atualizações de requisições para a rede
subnet	*Gauge*	*sub-rede*	*inst ID*	Duração da sub-rede
subnet.create	*Delta*	*sub-rede*	*inst ID*	Criação de requisições para a sub-rede
subnet.update	*Delta*	*sub-rede*	*inst ID*	Atualizações de requisições para a sub-rede
port	*Gauge*	*porta*	*inst ID*	Duração da porta
port.create	*Delta*	*porta*	*inst ID*	Criação de requisições para a porta
port.update	*Delta*	*porta*	*inst ID*	Atualizações de requisições para a porta
router	*Gauge*	*roteador*	*inst ID*	Duração do roteador
router.create	*Delta*	*roteador*	*instância*	Criação de requisições para o roteador
router.update	*Delta*	*roteador*	*inst ID*	Atualizações de requisições para o roteador
ip.floating	*Gauge*	*ip*	*iface ID*	Duração do IP flutuante
ip.floating.create	*Delta*	*ip*	*iface ID*	Criação de requisições para IP flutuante
ip.floating.update	*Delta*	*ip*	*iface ID*	Atualizações de requisições para IP flutuante

- **Métricas do *Glance***

  As métricas que podem ser coletadas no componente *Glance* são apresentadas na Tabela 3.8.

Tabela 3.8: Métricas do *Glance*

Nome	Tipo	Unidade	Recurso	Nota
image	*Gauge*	*imagem*	*imagem ID*	Polling de Imagem
image.size	*Gauge*	*B*	*imagem ID*	Tamanho da imagem carregada
image.update	*Delta*	*imagem*	*imagem ID*	Número de atualizações das imagens
image.upload	*Delta*	*imagem*	*imagem ID*	Número de imagens carregadas
image.delete	*Delta*	*imagem*	*imagem ID*	Número de imagens deletadas
image.download	*Delta*	*B*	*imagem ID*	Imagens transferidas
image.serve	*Delta*	*B*	*imagem ID*	Imagem fora do servidor

Essas métricas são importantes para analisar o comportamento da infraestrutura. Na próxima seção são descritos os conceitos da dependabilidade em relação à Computação em Nuvem. As métricas citadas anteriormente auxiliam na análise da dependabilidade dos sistemas computacionais. Monitorando os recursos computacionais da infraestrutura, é possível prever comportamentos que desviam o sistema da sua correta especificação e tomar decisões que previna esses comportamentos.

3.2 Dependabilidade

Essa seção apresenta os conceitos de dependabilidade, bem como a definição de seus atributos, de uma forma contextualizada ao ambiente de Computação em Nuvem. São tratados de uma maneira detalhada os atributos de disponibilidade e a confiabilidade. No decorrer do capítulo são discutidas as ameaças a dependabilidade, como falha, erro e defeito, e os meios de alcançar a dependabilidade. Para finalizar, são descritos vários modelos para avaliação de dependabilidade.

3.2.1 Conceitos Básicos

A dependabilidade de sistemas computacionais é fundamental não apenas para as empresas que utilizam sistemas de missão crítica, aqueles que necessitam de alta confiabilidade, por exemplo, o sistema de uma Usina Nuclear, mas também para usuários de escritório e para aqueles que utilizam os computadores de forma pessoal. Devido à dependência cada vez maior da sociedade em sistemas de *software*, os atributos de dependabilidade tornam-se cada dia mas importantes [Matias et al. 2013].

Todos os clientes têm uma expectativa cada vez maior da confiabilidade de seus produtos. A dependabilidade de um sistema pode ser definida como uma habilidade em realizar um serviço de qualidade, que é mensurada de acordo com cada atributo, para um usuário. Os usuários, por sua vez, são outros sistemas, físicos ou humanos, que interagem como a

interface do serviço.

Um sistema com dependabilidade, é aquele que não falha com frequência e, ao falhar, os seus dados não são perdidos [Dai et al. 2009]. A falha de um sistema pode ocorrer em virtude de um evento que desvia o serviço de seu comportamento esperado, ou seja, do seu correto funcionamento.

Provedores de Computação em Nuvem devem assegurar a continuidade dos negócios, garantindo a consistência e a disponibilidade do serviço prestado por suas plataformas de TI. Desse modo, tais plataformas são críticas, uma vez que uma falha em um seus componentes pode implicar grandes perdas financeiras diretas, ocasionadas por descumprimento de contratos, ou indiretas, ocasionadas por danos à reputação do provedor.

As aplicações desenvolvidas para a Computação em Nuvem devem ser confiáveis, ou seja, elas devem possuir uma arquitetura que permita que os dados permaneçam intactos mesmo que haja falhas ou erros em um ou mais servidores ou máquinas virtuais sobre os quais essas aplicações estão decompostas. Essa característica está associada à realização de cópias de segurança dos dados. O armazenamento dessas cópias deve ser feito em local seguro para que, caso haja alguma falha nas aplicações e elas percam os dados, esses ou pelo menos uma parte deles, possam ser recuperados.

3.3 Atributos da Dependabilidade

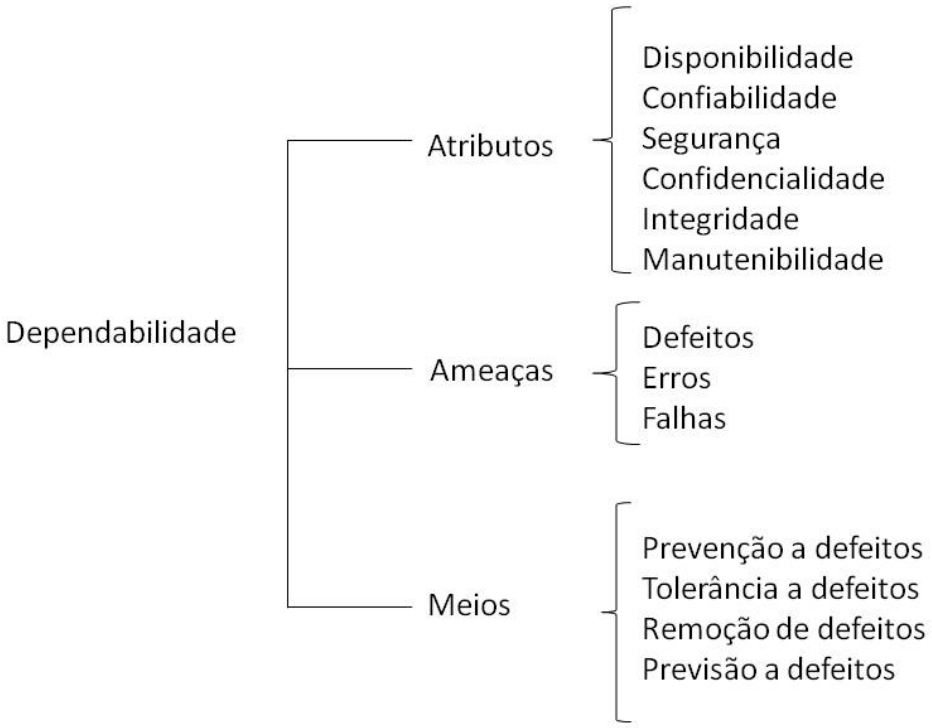
A Computação em Nuvem é um paradigma de computação distribuída em grande escala e suas aplicações são acessíveis em qualquer lugar e em qualquer momento. A alta dependabilidade de sistemas na nuvem torna-se importante e mas difícil de alcançar, devido à heterogeneidade dos equipamentos, do *software* e dos serviços. A dependabilidade de acordo com eles, é um conceito integrado que abrange os seguintes atributos [Avizienis et al. 2004] [Huang et al. 2013]:

- **Disponibilidade:** Habilidade em fornecer o serviço correto quando solicitado. Em outras palavras, a probabilidade de o sistema estar operacional quando solicitado.
- **Confiabilidade:** A continuidade para o serviço correto. Em outras palavras, a probabilidade de um defeito não ocorrer em um determinado período de tempo.
- **Segurança:** Ausência de consequências catastróficas para o usuário e o meio.
- **Confidencialidade:** O acesso aos serviços são limitados somente as entidades legítimas, ou seja, somente aquelas autorizadas.
- **Integridade:** O serviço não pode ser modificado sem autorização.
- **Manutenibilidade:** Disponível para receber modificações e reparos.

Esses atributos possibilitam a obtenção de medidas quantitativas, que são importantes para a monitoração do sistema. A partir das medidas coletadas, podemos analisar e, após a

análise, tomar decisões para prover um melhor serviço. A Figura 3.12 ilustra os atributos de dependabilidade, as ameaças ao sistemas e os meios de alcançar a dependabilidade [Huang et al. 2013].

Figura 3.12: Árvore de Dependabilidade



Fonte: Adaptado [Huang et al. 2013]

Uma das definições apresentadas para a dependabilidade, diz que ela é a "confiança no funcionamento", significa a confiança depositada no serviço a ser fornecido pelo sistema. A dependabilidade está relacionada diretamente com a *Quality of service* - Qualidade de Serviço (QoS), em como o usuário do sistema enxerga o comportamento dos serviços prestados [Veríssimo e de Lemos 1989]. A dependabilidade de um sistema pode ser avaliada sem perda de generalidade, baseando-se em duas medidas fundamentais: confiabilidade e disponibilidade [Muppala et al. 2000].

3.3.1 Confiabilidade

A confiabilidade é a probabilidade de que um sistema executará a sua função pretendida durante um período de tempo de funcionamento e em um ambiente determinado sem a presença de defeitos [Avizienis et al. 2004]. Logo, a confiabilidade do sistema pode ser

entendida como uma medida para a continuidade do serviço. Para descrever a confiabilidade de um dado sistema, é necessário conhecer a sua configuração, o estado em que ele é definido como operacional e as regras de operação [Kuo e Zuo 2003].

Para modelar a confiabilidade, é necessário levar em consideração os principais fatores que afetam a confiabilidade de um sistema: introdução de defeitos, remoção de defeitos e o ambiente em que o sistema opera [Musa et al. 1987]. Existem diversas formas de modelar a confiabilidade de um sistema, mas todas elas não são triviais, em virtude das características heterogêneas dos sistemas computacionais.

A teoria do cálculo das probabilidades permite obter uma quantificação da incerteza associada a um ou mais fatos e, portanto, é extremamente útil no auxílio de tomada de decisões [Barbetta et al. 2004]. Na teoria da probabilidade, a base de muitos resultados está relacionada com o estudo das variáveis aleatórias, suas funções densidade de probabilidade *(pdf)* e sua função de distribuição acumulada *(cdf)*. A probabilidade de que haja uma falha no decorrer do tempo t é dada por [Barbetta et al. 2004]:

$$F(t) = P[0 \leq T \leq t] = \int_0^t f(x)\,dx \tag{3.1}$$

onde F(t) é a função de distribuição acumulada da variável aleatória t.

A definição matemática do atributo de confiabilidade, é a probabilidade de que o sistema funcionará corretamente sem falhar no intervalo de tempo de 0 a t, é dada pela seguinte equação [Kuo e Zuo 2003]:

$$R(t) = P(T > t), t \geq 0 \tag{3.2}$$

onde T é uma variável aleatória, representando o tempo de falha ou tempo para falha.

Outras métricas utilizadas para confiabilidade são encontradas na literatura. Por exemplo, o *Mean Time to Failure* (MTTF), que é o tempo médio para falha. Esse parâmetro estima qual é o tempo médio de funcionamento de um sistema sem a presença de uma falha. O MTTF é definido por [Cannon et al. 2011]:

$$MTTF = \int_0^\infty R(t)\,dt \tag{3.3}$$

O *Mean Time Between Failures* (MTBF) que é o tempo médio entre falhas, essa métrica é utilizada para especificar quanto tempo um serviço funciona de forma correta antes de ocorrer uma falha, indica o tempo médio entre falhas no serviço. O MTBF é definido como [Cannon et al. 2011]:

$$MTBF = \frac{Tempo\ disponivel}{quantidade\ de\ falhas} \tag{3.4}$$

A outra métrica encontrada na literatura é denominada *Mean Time To Repair* (MTTR),

que é o tempo médio para reparo. Essa métrica se refere ao tempo médio para que um serviço volte a normalidade após a ocorrência de uma falha [Cannon et al. 2011].

$$MTTR = \frac{Tempo\ indisponivel}{quantidade\ de\ falhas} \tag{3.5}$$

3.3.2 Disponibilidade

Nos dias atuais, os usuários demandam por sistemas que atendam a qualquer hora do dia. Esses usuários acessam a sua conta bancária, realizam compras online e demais serviços, que têm de estar disponíveis sempre que necessitam deles. Para prover um ambiente com alta disponibilidade, todos os equipamentos devem passar por testes durante certo período de tempo. As métricas de disponibilidade podem ser obtidas por meio dos dados históricos de um sistema.

A disponibilidade pode ser definida como a prontidão de um sistema para entregá-lo corretamente um serviço em um dado momento [Avizienis et al. 2004]. A disponibilidade é a probabilidade de o seu sistema realizar uma função determinada em certo instante de tempo especificado, ou seja, ele se manter operacional em um certo período de tempo.

O *Uptime* representa o período de tempo que o sistemas se manteve operacional e o *Downtime* é o período de tempo em que o sistema se manteve indisponível devido a ocorrência de alguma falha ou uma manutenção. Matematicamente define disponibilidade da seguinte forma [Menasce]:

$$Disponibilidade = \frac{uptime}{uptime\ +\ downtime} \tag{3.6}$$

Um exemplo básico é se observar um sistema em um intervalo de tempo de dez horas. Suponha que o sistema tenha operado corretamente durante cinco horas (*uptime*) e no restante do tempo o sistema tenha sofrido algum tipo de falha. Conclui-se que a disponibilidade do sistema é de 50%.

3.4 Ameaças a Dependabilidade

As ameaças a dependabilidade são situações não esperadas, que levam o sistema a não prestar o serviço requisitado pelo usuário corretamente. As definições de falha, erro e defeito são encontradas na literatura de forma confusa e são importantes para o entendimento de todo o contexto do trabalho.

3.4.1 Falha

A Falha é um evento que ocorre quando a função realizada pelo sistema não está de acordo com a correta especificação que foi definida para a sua execução. Em outras

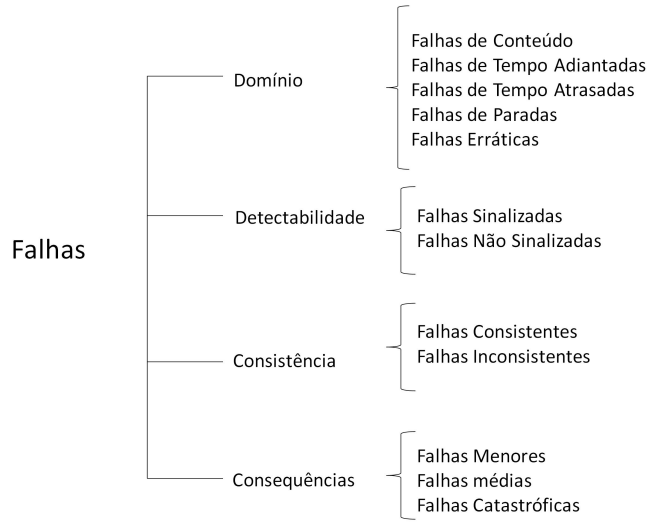
palavras, as falhas ocorrem quando os erros são propagados dentro do sistema. É de grande importância identificar as possíveis causas das falhas. Isso pode ser realizado mais facilmente, utilizando o modelo de classificação de falhas proposta nesta Dissertação de Mestrado. Os modos das falhas são as diferentes formas pelas quais os serviços se desviam da sua correta entrega. Esses modos são classificados de acordo quatro pontos de vista, o domínio, a detectabilidade, a consistência e as consequências [Avizienis et al. 2004].

Os modos das falhas são as diferentes formas pelas quais os serviços se desviam da sua correta entrega. Esses modos são classificados de acordo quatro pontos de vista [Avizienis et al. 2004]:

- Domínio de falha.
- Detecção de falha.
- Consistência da falha.
- Consequências das falhas no ambiente.

A Figura 3.13 mostra um resumo dos modos de falhas de serviços e suas classificações.

Figura 3.13: Modos de falhas de serviços



Fonte: Adaptado [Avizienis et al. 2004]

Os modos de falhas representam a forma pela qual as falhas são classificadas. As falhas apresentadas em um sistema são notadas após a sua ocorrência e então é realizada a classificação da sua consequência. Existem falhas que não são catastróficas se comparadas com outras, logo cada falha apresenta o seu nível de consequência. Com isso, pode-se afirmar que identificar e caracterizar um modo de falha é uma tarefa subjetiva.

3.4.2 Erro

O Erro é o desvio do serviço, entregue, em relação à sua correta especificação. O Erro caracteriza um estado incorreto de parte do sistema. Erros podem ocasionar falhas,

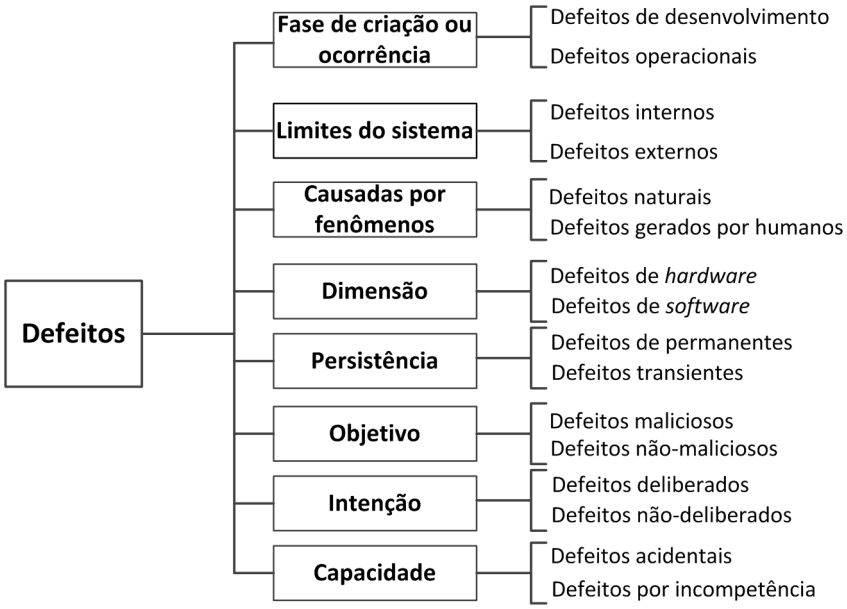
ao passo que as causas dos erros são os defeitos. As falhas surgem quando os erros são propagados no sistema. Salienta-se que a parte do sistema que contém erros pode nunca ser usada, dessa forma a falha poderá nunca ocorrer [Avizienis et al. 2004].

Os erros são classificados conforme a sua detectabilidade. Um erro pode ser detectável quando sua evidência é percebida pelo sistema. Já quando os erros estão presentes, mas não são notados, são chamados de erros latentes.

3.4.3 Defeito

O Defeito é a suposta causa de um erro. Os defeitos podem ser organizados e classificados de acordo com oito pontos de vista básicos [Avizienis et al. 2004]. Essa classificação leva à construção da classe fundamental de defeitos ilustrada pela Figura 3.14. Os defeitos são classificados de acordo a oito perspectivas básicas das classes elementares de defeitos que podem ocorrer durante toda vida útil de um sistema.

Figura 3.14: Classes elementares de defeitos



Fonte: Adaptado [Avizienis et al. 2004]

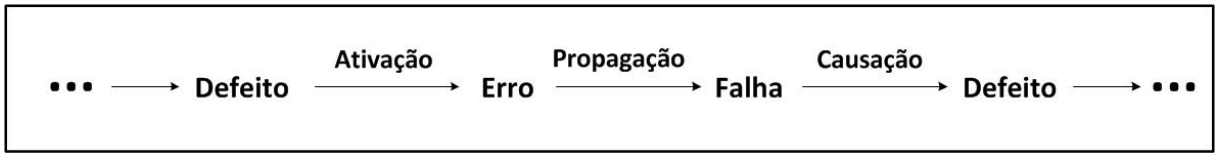
Se todas as combinações das oito classes elementares de defeitos fossem possíveis, haveria 256 diferentes classes de defeitos combinados, no entanto, nem todos os defeitos seguem o mesmo critério de classificação. Por exemplo, os defeitos naturais não podem ser classificados por objetivo e intenção. Dessa forma, foram identificadas 31 combinações possíveis. Essas combinações pertencem a três subgrupos:

- Defeitos de Desenvolvimento: incluem todas as classes de defeitos ocorridos durante o desenvolvimento.
- Defeitos Físicos: incluem todas as classes de defeitos que afetam o *hardware*.

- Defeitos de Interação: incluem todos os defeitos externos.

Esses três elementos possuem um relação causal denominada cadeia fundamental da dependabilidade. A Figura 3.15 apresenta um exemplo de encadeamento entre falha, erro e defeito [Avizienis et al. 2004].

Figura 3.15: Cadeia fundamental da dependabilidade



Fonte: Adaptado [Avizienis et al. 2004]

Após a ativação de um defeito, o erro pode propagar entre os diversos componentes do sistema antes de causar uma falha. A falha ocorrerá quando o erro for percebido pelo usuário como um desvio da função especificada. Existem algumas técnicas para se atribuir dependabilidade ao funcionamento de um sistema, descritas a seguir.

Para melhor compreensão dos conceitos de erro, falha e defeito, apresentamos um breve exemplo. Suponha-se que um determinado sistema esteja em funcionamento. Um usuário do sistema deseja enviar um pacote com dados. No momento do envio do pacote, um ruído eletromagnético ocorreu (defeito), modificando alguns *bits* do pacote. Devido à mudança dos *bits* (erro), o pacote não foi enviado. Consequentemente, o pacote não chegará ao destino final, conforme deveria ocorrer (falha).

3.5 Meios para Alcançar a Dependabilidade

Os meios para alcançar a dependabilidade são as ferramentas, métodos e soluções que possibilitam aos sistemas a capacidade de fornecer serviços com confiança. Nas últimas décadas, muitas técnicas foram desenvolvidas para alcançar os vários atributos de dependabilidade. Os meios para alcançar a dependabilidade estão definidos na Figura 3.12. Essas técnicas podem ser agrupadas em quatro grandes categorias, a seguir será detalhada cada categoria individualmente [Avizienis et al. 2004].

3.5.1 Prevenção a falhas

Prevenção a falhas é a prevenção de ocorrência ou introdução de algum tipo de falha no sistema. A prevenção a falhas se baseia na utilização de componentes e métodos robustos durante a construção do sistema ou plataforma, de modo a evitar a ocorrência de falhas durante o funcionamento. Entretanto, por melhores que sejam os componentes e as técnicas utilizadas, é impossível prevenir todas as falhas. Alguns sistemas podem ser extremamente complexos e de composição completamente heterogênea de maneira

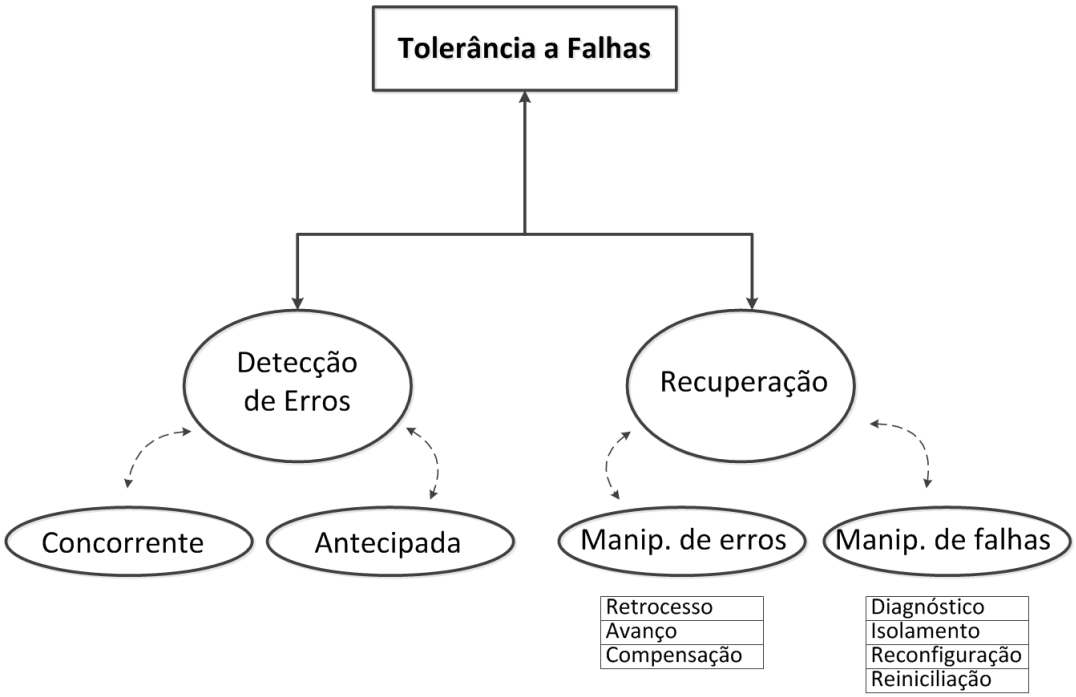
que prevenir a ocorrência de falhas se torna uma tarefa muito complicada. Além disso, eventos físicos externos podem levar o sistema a falhar [de Araújo Macêdo et al. 2010].

A prevenção a falhas pode ser utilizada durante as fases de especificações, evitando especificações incompletas ou confusas, no desenvolvimento escolhendo de forma correta as metodologias que utilizarão e definindo bem os processos, na fabricação verificando a qualidade dos componentes e durante a operação realizando periodicamente treinamento com os usuários do sistema. Com toda a evolução das técnicas de prevenção a falhas, na prática, é quase impossível garantir que um sistema não apresente falhas0 durante a sua operação.

3.5.2 Tolerância a falhas

Tolerância a falhas é o sistema fornecer o serviço esperado mesmo na presença de falhas. Para alcançar o objetivo, são utilizadas duas técnicas para a tolerância a falhas, a detecção de erros e a recuperação do sistema [Johnson 1989]. A Figura 3.16 apresenta os meios para obter a tolerância a falhas. A detecção de erros se baseia na detecção de um estado errado do sistema e ainda pode ser utilizada de maneira concorrente, durante a entrega normal do serviço ou de maneira antecipada, enquanto o serviço ainda está suspenso. Já os mecanismos de recuperação do sistema se baseiam na manipulação, na exclusão dos erros e na prevenção para que as falhas não sejam ativadas novamente.

Figura 3.16: Meios para obter tolerância a falhas



Fonte: [Johnson 1989]

A exclusão dos erros é composta de três partes, o retrocesso em que o sistema volta para o estado seguro; o avanço, em que o sistema é deslocado para um novo estado confiável e

a compensação em que são utilizados mecanismos de redundância para mascarar o erro. A manipulação de falhas é caracterizada por quatro funcionalidades, um diagnóstico em que é realizada uma identificação e localização dos erros, eliminação dos componentes que apresentarem alguma falha, reconfigurações no sistema e reinicialização.

A tolerância a falhas se baseia na utilização da redundância necessária para capacitar o sistema da habilidade de prover um serviço confiável mesmo após a ocorrência de uma falha. A construção de mecanismos de tolerância a falhas em sistemas distribuídos, como aqueles que se apresentam em ambientes de Computação em Nuvem, deve considerar aspectos básicos, que se relacionam não apenas com as hipóteses de falhas, mas também com os modelos de sincronia na interação entre os componentes do sistema, ou seja, síncronos (limites temporais conhecidos) e assíncronos (limites temporais desconhecidos). Os modelos tradicionais de sincronia para sistemas distribuídos são caracterizados por configurações homogêneas e estáticas em termos dos aspectos temporais, significando que, uma vez definidas, as características temporais dos componentes não muda ao longo do tempo [de Araújo Macêdo et al. 2010].

3.5.3 Remoção de falhas

A remoção de falhas é realizada durante a fase de desenvolvimento e a vida operacional do sistema, tem como objetivo principal reduzir o número de falhas no sistema e a sua gravidade. Logo, essa técnica busca alcançar a dependabilidade reduzindo o número de falhas durante as fases citadas anteriormente.

Na fase de desenvolvimento do sistema, são utilizados três procedimentos para a remoção de falhas [Johnson 1989]. Na fase de verificação, a cada nova funcionalidade adicionada ao sistema é realizada uma verificação para garantir que novas falhas não tenham sido introduzidos. Também pode ser realizado um diagnóstico que identifica e localiza as falhas existentes no sistema. E por último há o procedimento de correção das falhas localizados.

Já na fase de operação, a remoção de falhas pode ser realizada por meio da manutenção dos componentes. Existem dois tipos de manutenção, a corretiva e a preventiva. Na manutenção corretiva, a falha é removida após produzir defeitos no sistema. Já na manutenção preventiva, as falhas são removidos antes que ocasionem um defeito no sistema, a manutenção é realizada durante o funcionamento correto do sistema.

3.5.4 Previsão a falhas

A previsão a falhas tem a função de estimar o número atual, a incidência futura e as prováveis consequências das falhas ao sistema. Essa técnica realiza uma avaliação do comportamento do sistema em relação à ocorrência ou ativação de falhas [Johnson 1989]. As análises podem ser qualitativa ou quantitativa. Na análise qualitativa, elas são

identificas e classificadas baseadas nos defeitos que eles podem ocasionar ao sistema. No método quantitativo, métricas de dependabilidade são coletadas e a analisadas.

3.6 Modelos para Avaliação de Dependabilidade

Diversos modelos são utilizados para avaliar a dependabilidade de um sistema. Como exemplo, podem-se citar os Diagramas de Bloco de Confiabilidade e as Árvores de Falhas, que são os modelos combinatoriais mais utilizados. Também são utilizadas para modelar a dependabilidade de sistemas as Cadeias de Markov e as Redes de Petri Estocásticas, que são os modelos baseados em estados. Os modelos baseados em estados também podem ser chamados de não-combinatoriais. Cada modelo tem suas próprias características e funções individuais.

Um sistema é uma entidade que interage com outras entidades, ou seja, outros sistemas, como por exemplo, *hardware*, *software*, seres humanos e o mundo físico, com seus recursos e fenômenos naturais [Avizienis et al. 2004]. A seguir têm-se a descrição de alguns modelos encontrados na literatura para avaliar a dependabilidade do sistema. Na próxima seção cada modelo é descrito de forma sucinta, para que o leitor tenha um conhecimento breve de cada modelo.

3.6.1 Árvore de Falhas

A Análise de Árvores de Falhas (FTA) é utilizada para avaliar quantitativa e qualitativamente a confiabilidade e a disponibilidade de sistemas computacionais. Durante o desenvolvimento do sistema, pode-se realizar a análise qualitativa, para identificar potenciais problemas que podem ocorrer e ocasionar uma falha. Já a análise quantitativa serve para obter métricas de confiabilidade e disponibilidade do sistema. Os benefícios de uma árvore de falha são [Júnior et al. 1996]:

- Auxiliar a identificação dos modos de falha.
- Levantar os aspectos mais importantes do sistema para a falha de interesse.
- Auxílio gráfico para mudanças no sistema.
- Possibilidade de o analista concentrar em uma falha por vez.

As árvores de falhas são modelos gráficos que apresentam uma combinação de eventos responsáveis por conduzir uma falha no sistema. São compostas por portas lógicas e eventos; os eventos representam as condições normais e de falhas no sistema. Eles seguem uma lógica Booleana, em que ocorrem ou não ocorrem. Já as relações causa-efeito entre os eventos são representadas pelas portas lógicas.

Essa técnica é que foi utilizada nesta Dissertação de Mestrado. A escolha dessa técnica se deu por ser possível analisar quantitativamente as métricas desejadas de confiabilidade

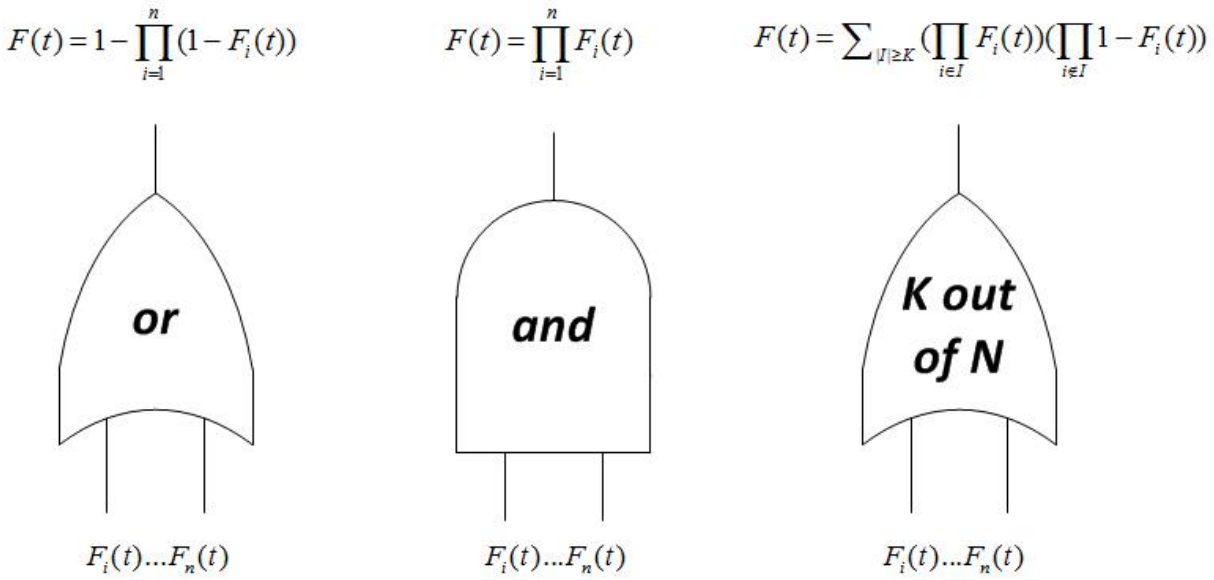
e disponibilidade, além dos benefícios citados anteriormente. As árvores de falhas podem ser classificadas de duas formas, coerentes e não coerentes. A seguir, uma descrição detalhada de cada classificação.

As coerentes são as árvores de falhas que não apresentam portas lógicas inversoras. Elas são classificadas em estáticas e dinâmicas. As estáticas são baseadas nas portas lógicas *and, or e k-out-of-n*. Logo, as sequências em que os eventos acontecem não são consideradas. Já na árvore dinâmica, há uma inclusão de novas portas lógicas como Porta de Dependência Funcional (FDEP), *Cold Spare (CSP)*, *Hot Spare (HSP)*, *Warm Spare (WSP)*, *Priority And* e Sequência Forçada (SEQ), com o objetivo de considerar a sequência em que os eventos acontecem. Já as portas lógicas inversoras são conhecidas como não coerentes.

O processo de criação de uma árvore de falha é dedutivo. Inicialmente, é realizada a definição do evento *Topo*, que representa uma falha no sistema. A partir do evento inicial, é possível conduzir uma análise retroativa e definir as causas da falha. A arquitetura da árvore ocorre em níveis, em que os eventos nos níveis inferiores são denominados de básicos e um evento que acontece mais de uma vez é chamado de evento repetido.

A avaliação quantitativa de uma árvore de falha é realizada calculando a probabilidade do evento *Topo* baseado nas probabilidades dos eventos básicos. O cálculo se difere pelo uso das portas lógicas, então, cada porta lógica utilizada tem o seu cálculo exclusivo. Se considerarmos n eventos distintos, em que a ocorrência do evento i é descrita pela sua função de distribuição acumulada (*cdf*) *Fi(t)*, as saídas das portas lógicas são definidas de acordo a Figura 3.17 [Rausand e Høyland 2004].

Figura 3.17: Função de distribuição acumulada das portas lógicas



Fonte: [Rausand e Høyland 2004]

A probabilidade de um evento do *Topo* é calculada utilizando as equações definidas na Figura 3.17, somente se a árvore de falha não apresentar eventos repetidos. Se isso acontecer, é necessário empregar técnicas diferentes.

3.6.2 Diagramas de Bloco de Confiabilidade

É umas das ferramentas mais comuns para análise de confiabilidade de sistemas. Diagramas de Blocos de Confiabilidade, conhecida como modelo *Reliability Block Diagram* - Diagramas de Bloco de Confiabilidade (RBD), define o relacionamento lógico entre componentes de um sistema [Kuo e Zuo 2003]. Os relacionamentos principais são: série, paralelo, ponte e *k-out-of-n*. Cada componente em um modelo RBD possui métricas de dependabilidade próprias. Uma das vantagens de utilizar este modelo é a facilidade de realizar a avaliação da confiabilidade e da disponibilidade [Xie et al. 2004].

O RBD é um modelo combinacional, sendo assim, possível realizar cálculos de confiabilidade e disponibilidade por meio de fórmulas fechadas (*closed forms*). Com a utilização dessas fórmulas a obtenção de resultados se torna mais rápida que a simulação [Trivedi et al. 1996].

Cada componente em um dado instante de tempo pode estar ativo, modo operacional, ou não, considerando a disponibilidade. A modelagem RBD pode ser considerada como um fluxo de sinais, da esquerda para a direita [Kuo e Zuo 2003]. Um componente ativo permite que o sinal percorra e um inativo impede a sua passagem. Logo no modelo RBD o sistema estará ativo, se, no mínimo, um caminho estiver ativo do início ao fim do sistema. Todas as métricas de dependabilidade são independentes.

Existem algumas premissas para a utilização dos modelos RBD como:

- A confiabilidade ou disponibilidade de cada bloco individualmente é conhecida ou estimada.
- As falhas dos blocos são estatisticamente independentes.
- Os blocos bimodais, ou seja, possuem apenas dois estados possíveis: operacional ou não.
- Todos os caminhos existentes são representados.

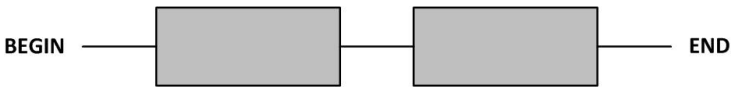
Caso todas as premissas sejam atendidas pelo sistema em teste, então é possível calcular as métricas de dependabilidade do sistema, utilizando o modelo RBD. Cada modelo RBD é detalhado a seguir.

3.6.3 Sistema em série

É a configuração mais comum no RBD. Os blocos de sistemas em série são ligados logicamente um seguido do outro. A Figura 3.18 ilustra esta forma de sistema. No modelo

de configuração em série, se um componente do sistema falhar, o sistema interrompe o serviço [Kuo e Zuo 2003].

Figura 3.18: Sistema em série



Fonte: [Kuo e Zuo 2003]

A confiabilidade de dois blocos em série, é definida pela Equação:

$$R_S(t) = R_1(t) \times \ R_2(t) \tag{3.7}$$

onde:

R1 é a confiabilidade do bloco 1.

R2 é a confiabilidade do bloco 2.

Considerando um sistema com n elementos, a confiabilidade do sistema de configuração em série é [Kuo e Zuo 2003]:

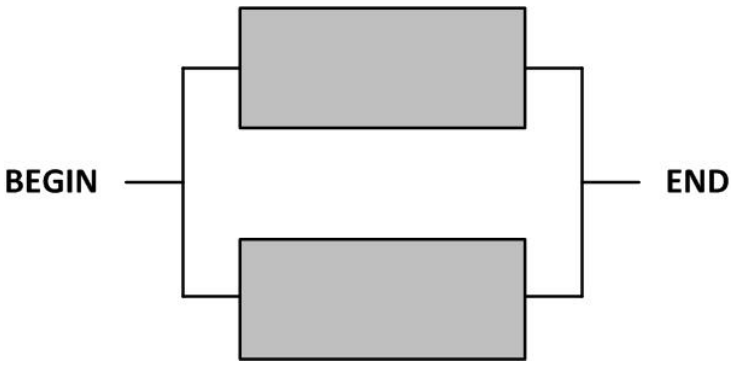
$$R_S(t) = \prod_{i=1}^{n} R_i(t) \tag{3.8}$$

onde *Ri(t)* é a confiabilidade do bloco *bi*.

3.6.4 Sistema em paralelo

Em uma configuração em paralelo, um sistema falha apenas quando todos os seus componentes também falham. A Figura 3.19 representa um sistema em paralelo [Kuo e Zuo 2003].

Figura 3.19: Sistema em paralelo



Fonte: [Kuo e Zuo 2003]

Em uma configuração em paralelo, considerando um sistema com n elementos, a con-

fiabilidade do sistema é obtida pela Equação [Kuo e Zuo 2003]:

$$R_p(t) = 1 - \prod_{i=1}^{n}(1 - R_i(t)) \tag{3.9}$$

onde *Ri(t)* é a confiabilidade do bloco *bi*.

Neste tipo de sistema pelo menos um componente deve estar em operação, para que todo o sistema esteja funcionando.

3.6.5 Cadeias de Markov

Cadeias de Markov são processos estocásticos com a chamada propriedade markoviana. Foram criados para realizar a modelagem de sistemas. É um modelo baseado em estado no qual se diz que o estado atual não depende dos estados anteriores para que se conheçam os estados seguintes. Por meio desse formalismo, é possível descrever o funcionamento de um sistema por meio de um conjunto de estados e transições.

Um processo estocástico é uma coleção de variáveis randômicas indexadas por elementos t pertencentes a um determinado intervalo de tempo. Um processo estocástico $\{X_n, n = 0, 1, 2...\}$, em que, X_n pode representar possíveis valores em um conjunto A finito ou infinito enumerável, chamado de conjunto de estados ou espaço de estados. O $X_n = 1$ representa um processo que está no estado i, no tempo n, logo X_n, é uma Cadeia de Markov ou possui a propriedade markoviana se [Haverkort 2002]:

$$P\left[X_{n+1} = j | X_n = i, X_{n-1} = i_{n-1}, ..., X_1 = i_1, X_0 = i_0\right] = P\left[X_{n+1} = j | X_n = i\right] = p_{ij} \tag{3.10}$$

Essa propriedade é interpretada da seguinte forma: a probabilidade condicional de qualquer estado futuro independe dos estados passados, ou seja, para conhecer o estado futuro é preciso somente conhecer o estado atual. As probabilidades podem ser representadas por uma matriz de transição :

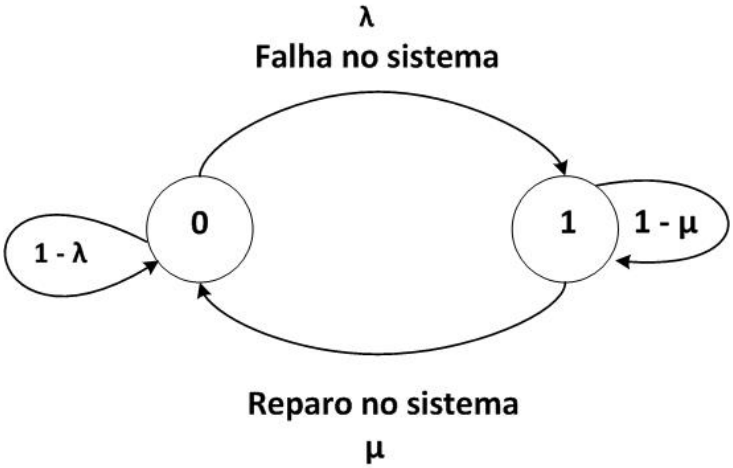
$$\begin{pmatrix} p_{0,0} & p_{0,1} & p_{0,2} & \cdots \\ p_{1,0} & p_{1,1} & p_{1,2} & \cdots \\ \vdots & \vdots & \vdots & \ddots \\ p_{i,0} & p_{i,1} & p_{1,2} & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix} \tag{3.11}$$

Na Cadeia de Markov, o tempo pode assumir valores contínuos ou discretos. As Cadeias de Markov em tempo discreto são conhecidas como *Discrete-time Markov Chains* (DTMC), as transições acontecem em tempos discretos. Já a de tempo contínuo, chamadas de *Continuous-time Markov Chains* (CTMC) as suas transições podem ocorrer em

qualquer instante do tempo [Haverkort 2002]. A representação da Cadeia de Markov pode ser realizada pela máquina de estados, em que as setas representam as possíveis transições com os valores das probabilidades e os círculos as transições entre os estados do modelo.

Para o melhor entendimento da Cadeia de Markov, tem-se a descrição de um exemplo com dois estados S_1 e S_2, conforme ilustra a Figura 3.20.

Figura 3.20: Exemplo Cadeia de Markov com dois estados



Fonte: [Haverkort 2002]

O sistema pode está funcionando em seu estado normal (0) ou pode apresentar uma falha (1) no decorrer de um tempo. O estado de falha continua até que o sistema seja reparado, após ser reparado sofre a transição para o estado normal.

A matriz Q é dada por [Haverkort 2002]:

$$\begin{pmatrix} 1-\lambda & \lambda \\ \mu & 1-\mu \end{pmatrix} \tag{3.12}$$

A probabilidade de falha em $t + dt$, dado que o sistema esteja trabalhando no tempo t [Haverkort 2002]:

$$P\left(0|0\right) \equiv P\left[x(t+dt) = 0|x(t) = 0)\right] = 1 - \mu dt \tag{3.13}$$

$$P\left(0|1\right) \equiv P\left[x\left(t+dt\right) = 0|x(t) = 1\right] = \lambda dt \tag{3.14}$$

$$P\left(1|0\right) \equiv P\left[x\left(t+dt\right) = 1|x(t) = 0\right] = \lambda dt \tag{3.15}$$

$$P\left(1|1\right) \equiv P\left[x\left(t+dt\right) = 1|x(t) = 1\right] = 1 - \lambda dt \tag{3.16}$$

3.6.6 Redes de Petri Estocásticas

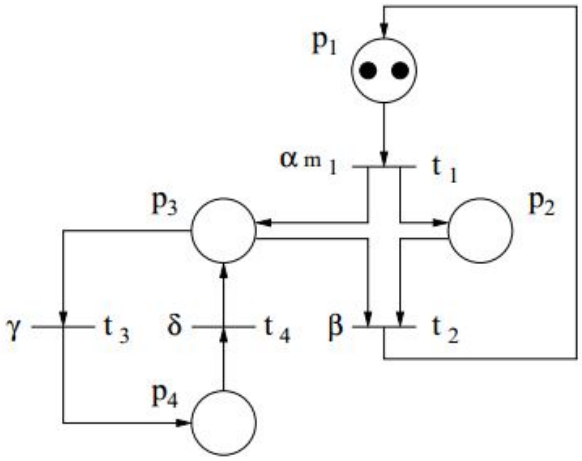
Diversos modelos de dependabilidade utilizam as Redes de Petri para realizar a modelagem da dependabilidade dos sistemas. As Redes de Petri são compostas por dois elementos, o ativo, que é chamado de transição e o outro passivo, conhecido como lugar. Logo, as transições representam as ações e os lugares representam as variáveis. A Figura 3.21 apresenta os elementos básicos de um grafo utilizando Redes de Petri [Francês 2003].

Figura 3.21: Grafo e seus elementos básicos

Fonte: [Francês 2003]

O modelo *Stochastic Petri Net* - Redes de Petri Estocásticas (SPN) é baseado nos conceitos de Redes de Petri, é muito utilizado na análise de dependabilidade de sistemas. A SNP foi criado para a modelagem estocástica aplicada, o seu objetivo principal é fornecer um método que permita a integração da descrição formal, prova de correção e avaliação de desempenho de sistemas envolvendo concorrência, não-determinismo e sincronização. A diferença entre um modelo SPN e uma Rede de Petri usual são as transições. As transições habilitadas na SPN são realizadas com um atraso exponencialmente distribuído [Haas e Haas]. Diferentemente da Rede de Petri normal, que a transição habilitada é executada no mesmo momento. A Figura 3.22 ilustra um exemplo de SPN.

Figura 3.22: Exemplo de uma Rede de Petri Estocástica



Fonte: [Haas e Haas]

As avaliações de desempenho dos sistemas descritos por uma SPN podem ser realizadas por Cadeias de Markov de Tempo Contínuo, por conta do método probabilístico

que transforma o determinismo de uma rede não temporizada em probabilidade na rede temporizada. Um estado em uma CTMC corresponde a uma marcação em uma SPN, e a transição de uma CTMC corresponde ao disparo de uma transição de uma SPN. A SPN é definida como uma 6-upla SPN=(P,T,A,PA,MO,D), em que:

- **P:** Conjunto de lugares
- **T:** Conjunto de transições
- **A:** Conjunto de arcos
- **PA:** Pesos dos arcos
- **Mo:** Marcação inicial
- **D:** Conjunto das taxas de disparos associadas às transições que obedecem a uma distribuição exponencial.

Utilizando um formalismo, pode-se dizer que as SPNs são Redes de Petri com um conjunto adicional de taxas, uma para cada transição, que definem as *pdfs* das exponenciais. As SPNs funcionam da seguinte forma: quando uma transição é habilitada, uma amostra da sua distribuição é atribuída com um determinado tempo. A transição é disparada após o tempo acabar, a solução de conflitos entre transições se dá com o disparo daquela que tiver o menor tempo para ser disparada.

3.7 Conclusão

Neste capítulo foram apresentadas as plataformas de gerenciamento para nuvem. Mostrou-se a importância de se ter uma plataforma de gerenciamento em uma infraestrutura de Computação em Nuvem. Em seguida, foram discutidos conceitos de dependabilidade em relação à Computação em Nuvem. Além disso, os atributos, ameaças e meios de alcançar a dependabilidade foram detalhados.

É mostrada também a necessidade de se ter um ambiente de Computação em Nuvem monitorado durante a sua vida útil para conseguir prevenir, tolerar, remover e prever as possíveis falhas no sistema. Diversos modelos de dependabilidade podem ser utilizados para modelar os dois principais atributos da dependabilidade do sistema: confiabilidade e disponibilidade.

Capítulo 4

A ArqDep

Uma arquitetura pode ser definida como um conjunto de estruturas que formam um sistema, suas relações e suas propriedades. As propriedades são os requisitos não funcionais como escalabilidade, desempenho, segurança, elasticidade, disponibilidade, monitoramento e outros. Esta Dissertação de Mestrado tem como objetivo principal especificar e implementar uma arquitetura de Computação em Nuvem. Essa arquitetura apresenta os conceitos de dependabilidade.

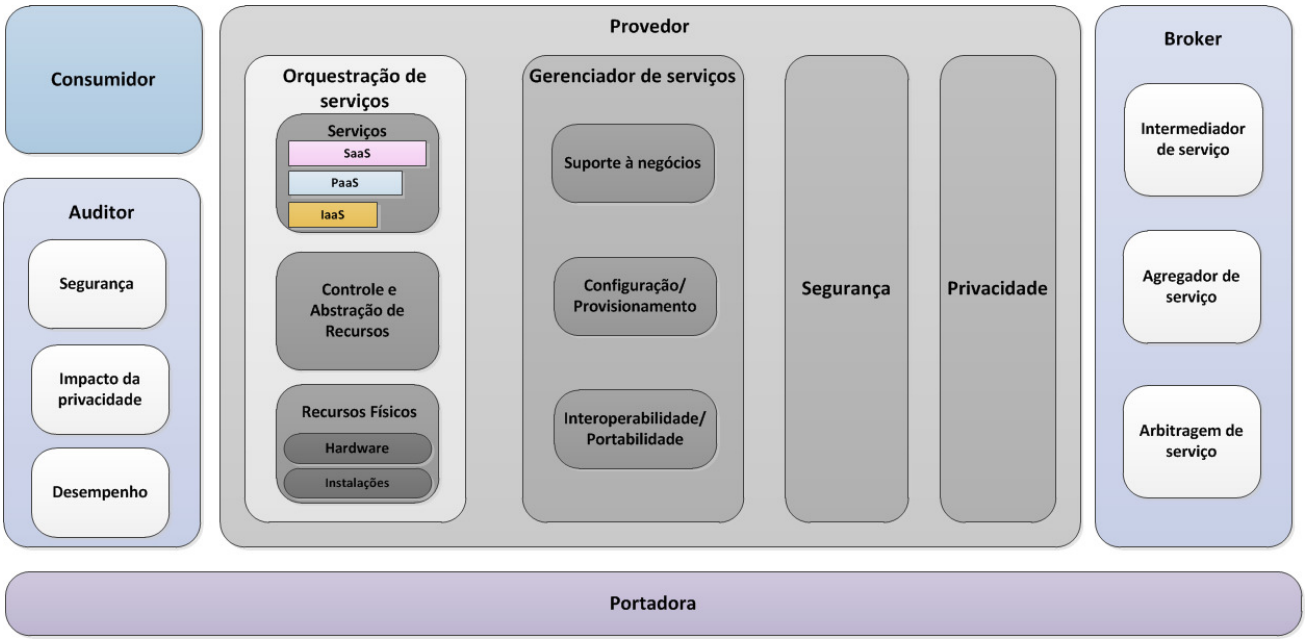
A ArqDep é uma arquitetura para a Computação em Nuvem, que utiliza alguns componentes da arquitetura de referência do NIST [Liu et al. 2011] ilustrada na Figura 4.1 e do gerenciador de infraestrutura *OpenStack* [OpenStack 2013a]. O NIST desenvolveu o modelo de referência para auxiliar no desenvolvimento de arquiteturas de Computação em Nuvem. Esse modelo de referência foi desenvolvido baseado num conjunto de estudos de modelos de referência existentes e propostos por empresas e centros de pesquisas que desenvolvem serviços de Computação em Nuvem [Liu et al. 2011].

A arquitetura de referência do NIST apresenta cinco atores principais: o Consumidor que é aquele que utiliza os serviços prestados pelo Provedor; o Provedor, que realiza a entrega dos serviços aos Consumidores; o Auditor que realiza uma auditoria se os serviços prestados estão de acordo com o que foi acordado no SLA em relação a desempenho, segurança e privacidade. O *Broker* que é o intermediador entre o Provedor e o Consumidor e, por último, a Portadora que é usada como um canal de comunicação entre todos os atores.

Já o *OpenStack* é um gerenciador de infraestrutura virtual de código aberto que permite a criação de nuvens IaaS públicas e privadas, com suporte a diversos *hypervisors*. O desenvolvimento desse sistema foi iniciado em 2010, por uma parceria entre a NASA e o Provedor de nuvem *Rackspace*; o *OpenStack* permite a implementação de nuvens simples e massivamente escaláveis.

A versão *Havana*, é a última versão do *OpenStack*, tem como objetivo auxiliar na construção de infraestruturas para prover serviços na nuvem para organizações ou pessoas que utilizam serviços nas nuvens. A Figura 4.2 mostra a arquitetura lógica completa do

Figura 4.1: Arquitetura de referência do NIST



Fonte: [Liu et al. 2011]

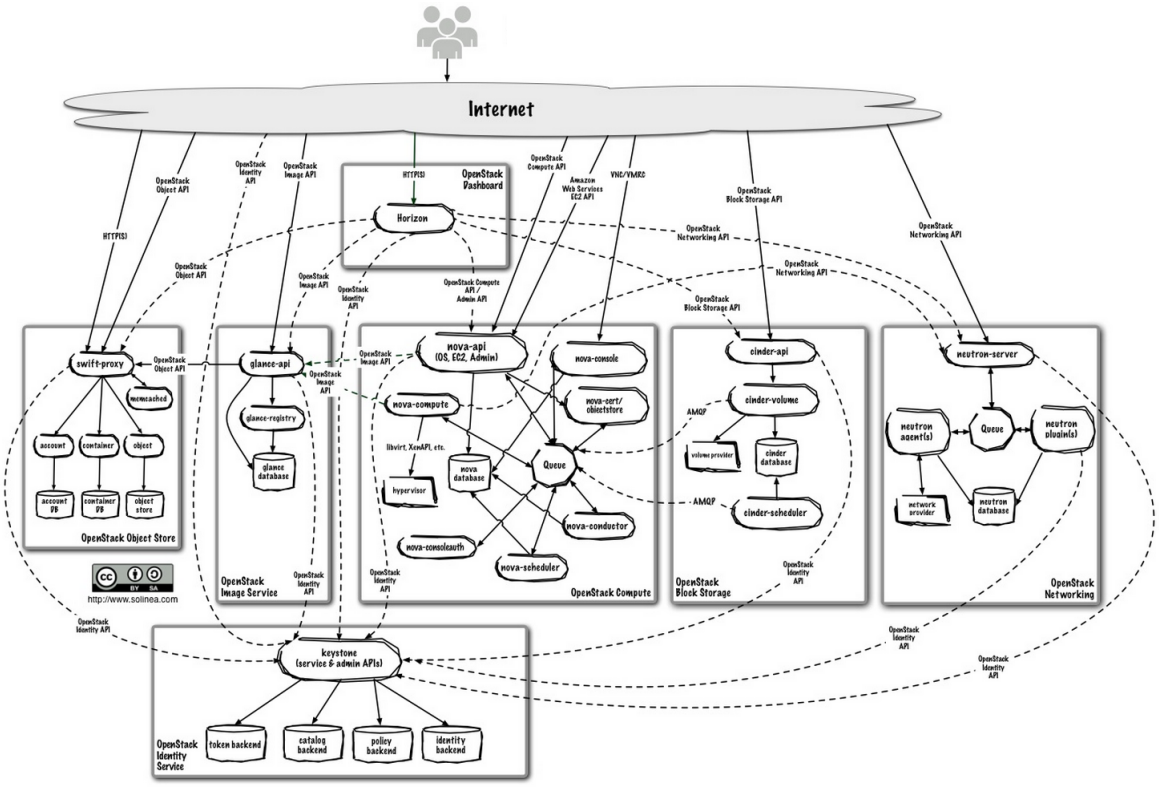
OpenStack na versão *Havana*. O *OpenStack* tem a função parecida com a de um sistema operacional. Assim como o sistema operacional gerencia os componentes de um computador, o *OpenStack* gerencia todos os componentes operacionais da nuvem. Portanto, a utilização do *OpenStack* facilita o controle dos recursos disponíveis na infraestrutura.

Na ArqDep, uma nova camada chamada de Dependabilidade e diversos módulos são criados. Essa arquitetura permite oferecer serviços aos Consumidores realizando o monitoramento dos recursos computacionais do sistema, que afeta a disponibilidade e a confiabilidade. Durante a utilização dos serviços prestados pelo Provedor, tanto o administrador da infraestrutura quanto os Consumidores poderão acessar os dados de monitoramento coletados na infraestrutura pelo componente *Ceilometer*, que é um componente do *OpenStack* responsável em coletar informações e métricas a respeito dos demais componentes instalados na infraestrutura.

Devido à necessidade de se ter ambientes cada dia mais confiáveis, surgiu o conceito de dependabilidade. A dependabilidade de um sistema pode ser entendida como a capacidade de oferecer uma funcionalidade específica, que pode ser justificadamente confiável [Avizienis et al. 2004], ou ainda, que o sistema executará ações especificadas ou apresentar resultados específicos de maneira confiável [Parhami 1988]. Esse conceito fez com que a *International Organization for Standardization (ISO)* elaborasse a regulamentação ISO 9000-4/1993 com as Normas de Gestão da Qualidade e Garantia da Qualidade Parte 4: Guia para a gestão do programa de dependabilidade [ISO 2013].

Essa regulamentação consiste em um guia para gestão do programa de dependabilidade de produtos e serviços. É uma norma técnica que trata dos conceitos de dependabilidade,

Figura 4.2: Arquitetura lógica do *OpenStack*



Fonte: [OpenStack 2013a]

sendo dividida em seis seções. As quatro primeiras seções abordam os objetivos e a terminologia da norma, além de diretrizes para a organização e o estabelecimento de responsabilidades em um programa de dependabilidade. As seções finais detalham os elementos de programas de dependabilidade genéricos e específicos de projetos ou de produto.

O que diferencia a ArqDep das outras arquiteturas, já presentes no mercado, é a forma pela qual as métricas dos recursos computacionais são disponibilizadas aos Consumidores dos serviços. Nesta arquitetura, diferentemente das outras, são disponibilizados valores absolutos de disponibilidade, sendo assim, os Consumidores poderão verificar se a disponibilidade dos serviços está de acordo com o que foi assinado no SLA. Os administradores da infraestrutura têm acesso a métricas que auxiliam na tomada de decisões. Também é realizado pela ArqDep um gerenciamento da infraestrutura, por meio de um monitoramento e controle.

A disponibilidade de todos os nós e instâncias configuradas na infraestrutura pode ser verificada utilizando alguns comandos do sistema operacional. Por exemplo, pode-se utilizar o PING. Este comando será executado em tempos pré-determinados. O PING utiliza-se de um protocolo chamado *Internet Control Message Protocol (ICMP)*, um protocolo de controle, que utiliza dois mecanismos: um de requisição e um de resposta. O mecanismo de requisição é chamado de *echo_ request* e o mecanismo de resposta é conhe-

cido como *echo_ reply*.

O monitoramento realizado por essa arquitetura auxiliará tanto os Consumidores quanto os administradores da infraestrutura a verificar se os serviços estão de acordo com o SLA. Os administradores da infraestrutura são capazes de tomar decisões, baseando-se nos dados do monitoramento. Os usuários dos serviços do Provedor de Computação em Nuvem terão acesso a diversas métricas que são coletadas pela ArqDep.

Conforme mencionado anteriormente, os atores do NIST presentes na arquitetura proposta são: Provedor, Consumidor e Portadora. O Provedor que realiza a entrega dos serviços aos Consumidores. O Consumidor que é aquele que utiliza os serviços prestados pelo Provedor e a Portadora que é o responsável por prover a comunicação entre todos os atores.

O Provedor apresenta três camadas: Orquestração, Controlador e Dependabilidade. A camada de Orquestração é responsável em definir a forma pela qual os serviços serão prestados aos Consumidores. Essa camada também é responsável pela abstração dos recursos na infraestrutura e pelos recursos físicos.

A camada Controlador é responsável por todos os componentes utilizados para a prestação dos serviços de Computação em Nuvem. E, por último, tem-se a camada de Dependabilidade, que permite o monitoramento de diversas métricas que afetam a disponibilidade e a confiabilidade dos sistemas de Computação em Nuvem. Logo, o gerenciamento dos recursos de todos dos componentes do sistema é realizado por essa camada.

4.1 Visão Geral

As arquiteturas dos diversos sistemas de Computação em Nuvem diferem entre si, dependendo da política adotada pelo arquiteto e pelo desenvolvedor da arquitetura. A ArqDep é uma arquitetura que utiliza os componentes da Arquitetura de Referência do NIST [Liu et al. 2011], integrando os aspectos de análise da dependabilidade e é implementada utilizando o gerenciador de nuvem *OpenStack*.

Em sistemas complexos, como o de Computação em Nuvem, é necessário prover um ambiente seguro e confiável aos usuários, logo os atributos de dependabilidade tornam-se necessários para o ambiente. A análise de dependabilidade é uma atividade essencial que visa a fornecer meios para que seja possível promover melhorias na qualidade dos serviços prestados.

A Computação em Nuvem é um paradigma de computação distribuída em grande escala e suas aplicações são acessíveis em qualquer lugar e em qualquer momento. Assim sendo, sistemas de Computação em Nuvem dependáveis tornam-se importantes e, em decorrência da heterogeneidade de seus componentes, tornam-se mas difíceis de alcançar [Trivedi et al. 1996].

Existem vários motivos pelos quais é necessário o monitoramento de um ambiente de

Computação em Nuvem. Logo, construir uma infraestrutura que realize o monitoramento completo do ambiente de Computação em Nuvem que atenda a todos os motivos, é um desafio cada vez maior. Os principais motivos para se ter um ambiente de monitoramento são [Elmroth e Larsson 2009]:

- Garantir que as máquinas virtuais obtenham a capacidade estipulada no SLA.
- Coletar dados para contabilizar os recursos utilizados. A bilhetagem é realizada por meio desse monitoramento.
- Medições da infraestrutura, por exemplo, memória utilizada, memória livre e largura de banda.
- Indicadores de desempenho.

Todos os motivos citados acima são importantes para utilizar uma arquitetura que realize o monitoramento da infraestrutura que prover serviços em nuvem. A ArqDep realiza o monitoramento de todos os recursos computacionais da infraestrutura. A arquitetura foi implantada em uma IaaS privada. O modelo de implantação em nuvem privada e IaaS foram escolhidos para implementação da arquitetura porque essa combinação fornece um maior controle para a organização que utiliza os seus serviços [Liu et al. 2011].

O modelo de serviço IaaS é um dos modelos mais importantes sob os aspectos de pesquisa e comercial, pois pode trazer grandes avanços na diversidade de prestação de serviços em nuvem. A utilização desse modelo pode acarretar grandes benefícios às pequenas e médias empresas, laboratórios de pesquisas e departamentos governamentais, aumentando o aproveitamento dos equipamentos e criando um ambiente corporativo e colaborativo de alta tecnologia.

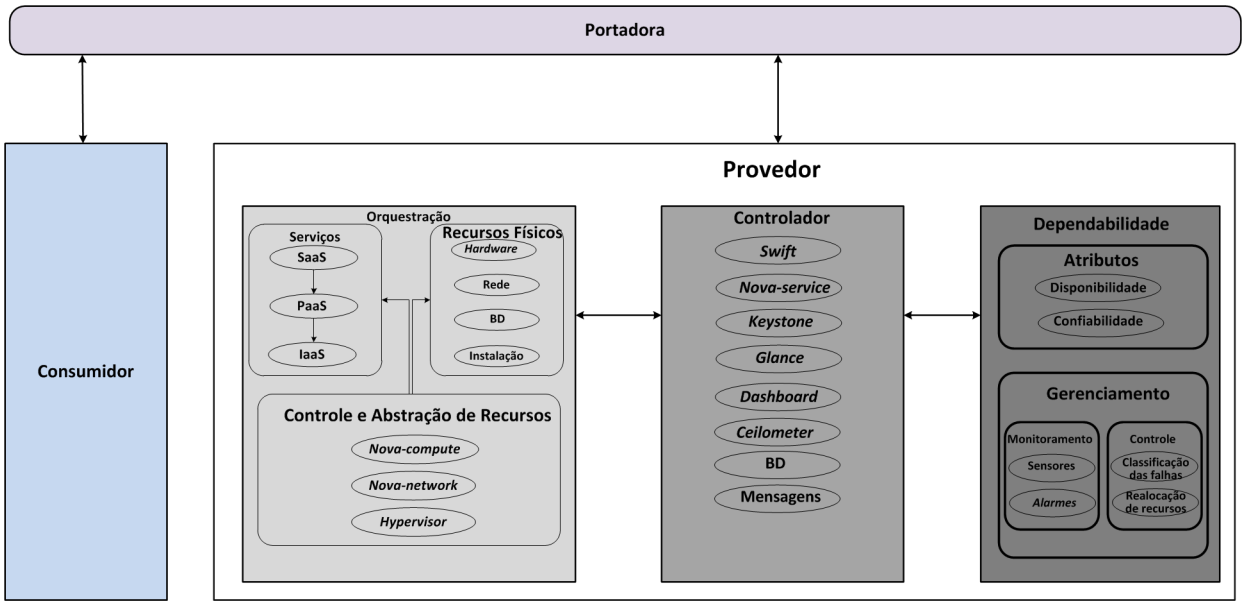
Na arquitetura de referência do NIST, existem cinco atores principais: Consumidor, Auditor, Provedor, *Broker* e Portadora. A ArqDep diferencia pelo número de atores principais e pela adição de camadas e módulos. Uma nova camada denominada de Dependabilidade de serviços é incluída no ator Provedor, essa camada tem como função realizar o gerenciamento da infraestrutura, por meio do monitoramento contínuo do sistema. Esse monitoramento é realizado pelo novo projeto do *OpenStack*, denominado de *Ceilometer*. O *Ceilometer* permite coletar dados de medição da nuvem privada de forma centralizada, obtendo assim, estatísticas mais detalhadas.

4.2 Descrição da ArqDep

Todos os componentes da ArqDep e suas respectivas funcionalidades estão bem definidos e divididos, o que permite a dependabilidade, a simplicidade e a eficiência para o provimento de serviços em nuvem. A Figura 4.3 ilustra a arquitetura proposta, chamada de ArqDep.

A ArqDep tem como função oferecer aos seus usuários serviços de Computação em Nuvem de forma simples, com recursos computacionais escaláveis, com um controle dos recursos computacionais, por meio de um monitoramento contínuo dos recursos computacionais e uma classificação das possíveis falhas apresentadas na infraestrutura. Os conceitos de dependabilidade na arquitetura podem ser vistos nas características apresentadas de monitoramento dos recursos e classificação das possíveis falhas na infraestrutura do Provedor de serviços em nuvem.

Figura 4.3: A ArqDep



Fonte: Autora

A ArqDep contém três atores principais como podem serem vistos na Figura 4.3. O primeiro ator é o Consumidor, o segundo é o Provedor com três camadas, e o último ator é a Portadora. O ator Provedor apresenta três camadas e diversos módulos. A seguir é detalhada a função de cada ator principal da ArqDep, bem como a função de cada camada e dos módulos presentes.

4.2.1 As relações entre os atores da ArqDep

A ArqDep apresenta três atores principais, diferentemente da arquitetura de referência do NIST que contém cinco atores principais. Os atores *Broker* e Auditor foram excluídos, em relação à arquitetura do NIST, pois estes atores não comprometem as funcionalidades da arquitetura. Em um trabalho futuro, pretende-se utilizar estes atores para um efeito comparativo. Os atores que foram mantidos são o Consumidor, o Provedor e a Portadora. Cada ator tem uma relação com os demais atores e suas funções. A seguir, uma descrição da relação de cada ator.

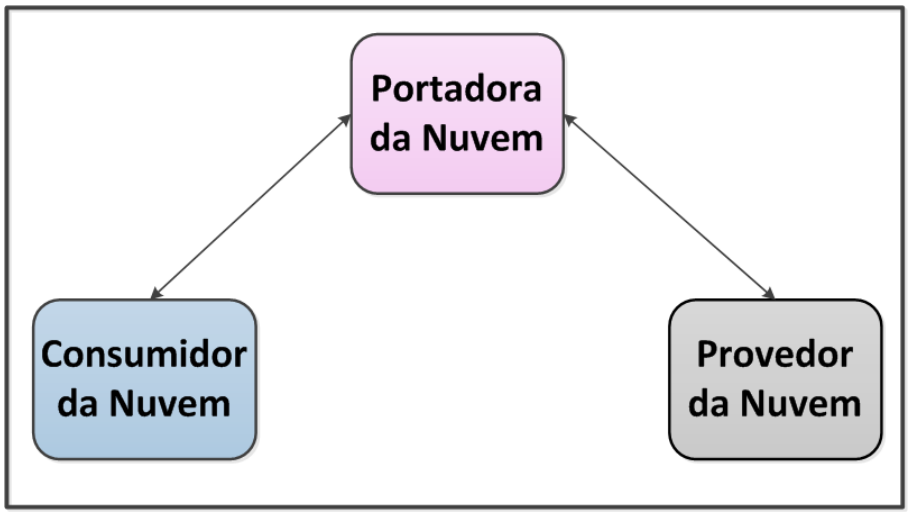
O Consumidor é uma pessoa ou organização que mantém uma relação de negócios e usa o serviço do Provedor da nuvem. Este ator é o que utilizará todos os serviços

hospedados na infraestrutura de nuvem. Uma infraestrutura em nuvem podem conter diversos Consumidores diferentes. Os serviços são prestados a esses Consumidores de acordo com o que foi assinado no SLA. Se por algum motivo o Provedor de serviços de Computação em Nuvem ou os usuários infringirem com o que está acordado no SLA, isso pode acarretar em punições para ambas as partes.

O Provedor é uma pessoa, organização ou entidade responsável por fazer um serviço disponível para os Consumidores da nuvem. O Provedor mantém e controla a infraestrutura da nuvem para oferecer a seus Consumidores serviços em nuvem de forma transparente, oferecendo recursos computacionais sob demanda a seus Consumidores. Toda a manutenção e o gerenciamento da infraestrutura são de responsabilidade do Provedor de serviços de Computação em Nuvem. Os Provedores de serviços de Computação em Nuvem têm a responsabilidade de fornecer um ambiente seguro, em que os dados dos Consumidores não sofram nenhum tipo de violação, disponibilidade dos seus serviços, escalabilidade e confiabilidade.

A Portadora faz o papel do intermediário que fornece conectividade e transporte entre Provedor e Consumidor de serviços nas nuvens. Na nossa arquitetura, a Portadora é a *Internet*, por meio da qual todos os serviços da infraestrutura serão disponibilizados. Sendo assim, para acessar os recursos oferecidos pelos Provedores é necessário, somente um dispositivo que tenha acesso à *Internet* e um enlace de dados confiável para a utilização dos serviços. O Consumidor só consegue se comunicar ao Provedor via Portadora. A Figura 4.4 mostra as relações entre os atores da arquitetura de Computação em Nuvem.

Figura 4.4: Relações entre atores da arquitetura



Fonte: Autora

O Consumidor e o Provedor só se comunicam utilizando a Portadora. Todas as requisições de serviços realizadas pelos Consumidores são enviadas por meio da Portadora e os serviços são entregues pelo Provedor também via Portadora. Como já citado anteriormente, a Portadora é a *Internet*. O Provedor de serviços de Computação em Nuvem

podem criar no SLA regras para contratos de serviços de comunicação, em que o fornecedor dos serviços de telecomunicações contratados pelos usuários, garanta níveis confiáveis nas conexões, pois é necessário possuir uma conexão de qualidade e segura, entre os Consumidores e os Provedores de serviços de Computação em Nuvem.

4.2.2 As relações entre as camadas do Provedor

O Provedor apresenta três camadas: a Orquestração, o Controlador e a Dependabilidade. A camada de Orquestração é responsável pela forma de a infraestrutura oferecer os seus diversos serviços em nuvem, para atender às diferentes necessidades de negócios e TI.

Essa camada também é responsável por orquestrar os serviços providos, realizar a abstração dos recursos, por exemplo, a criação de novas máquinas virtuais e pelo gerenciamento dos recursos físicos da infraestrutura. A camada de Orquestração apresenta três módulos: o de serviços, o de recursos físicos e o de abstração de recursos. Cada módulo contém as suas próprias funcionalidades.

O módulo de serviços é onde o Provedor de serviços em nuvem, define as interfaces que os Consumidores dos serviços acessarão os serviços de Computação em Nuvem. O módulo de abstração de recursos contém todos os componentes do sistema que os Provedores de serviços em nuvem utilizam para gerenciar e fornecer os recursos de Computação em Nuvem. O último módulo é o de recursos físicos que contém todos os recursos físicos necessários para a infraestrutura de Computação em Nuvem.

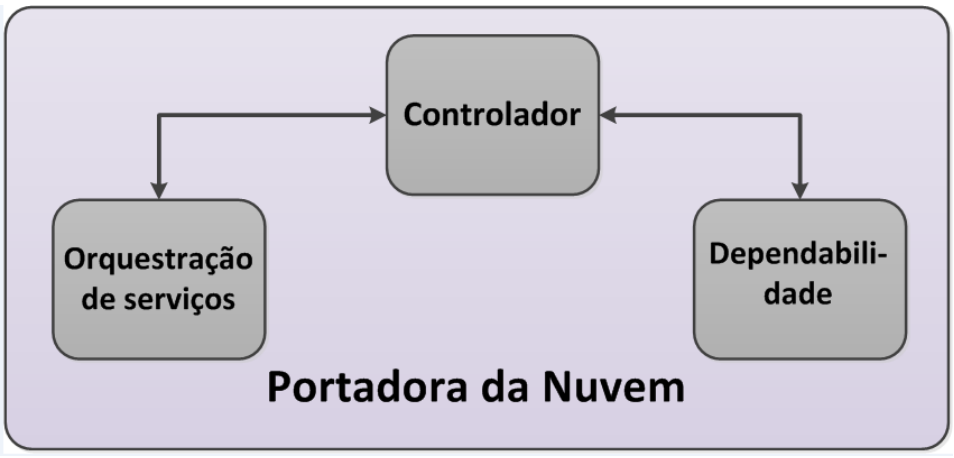
A camada Controlador inclui todas as funções para a operação dos serviços prestados aos Consumidores. Todos os componentes para a prestação dos serviços pelo Provedor estão no Controlador. De forma geral, o Controlador executa o controle dos serviços prestados pelo Provedor aos Consumidores. No Controlador encontram-se os módulos *Swift*, *Nova-service*, *Keystone*, *Glance*, *Dashboard*, *Ceilometer*, Banco de dados e o responsável pela troca de mensagens. Esses componentes são os responsáveis por fornecer recursos computacionais aos usuários da infraestrutura.

A camada de Dependabilidade tem como função realizar o gerenciamento dos serviços fornecidos pelo Provedor de nuvem aos seus Consumidores. Todos os recursos computacionais do sistema, como memória, disco, CPU, disponibilidade das instâncias e outras métricas que afetam a confiabilidade do sistema devem ser monitorados por essa camada. Para realizar a coleta e análise dos dados é utilizado o componente do *Openstack* conhecido como *Ceilometer*. A camada de Dependabilidade apresenta dois módulos: o de atributos e o de gerenciamento.

A camada Dependabilidade também é responsável por realizar o controle do sistema, realizando a classificação das falhas encontradas na infraestrutura; essa classificação se baseia no modelo que será proposto nesta Dissertação de Mestrado e de realizar a realocação

de recursos quando necessário. Todas as comunicações entre as camadas do Provedor são realizadas utilizando a Portadora. A Figura 4.5 ilustra as relações entre as camadas do Provedor de nuvem.

Figura 4.5: Relações entre as camadas do Provedor



Fonte: Autora

O Controlador está relacionado com a Orquestração de serviços, controlando todos os serviços prestados aos Consumidores e, ao mesmo tempo, relaciona-se com a Dependabilidade. A Dependabilidade realiza o gerenciamento da infraestrutura, com a função de monitoramento e controle das métricas que afetam a confiabilidade e disponibilidade do sistema. O componente *Ceilometer* realiza o monitoramento e coleta das métricas geradas pelos componentes do Controlador e disponibiliza para os Consumidores e administradores da infraestrutura. No monitoramento é possível acoplar *plugins* para coletar métricas específicas e incluir alarmes na infraestrutura.

4.2.3 As camadas do Provedor

As três camadas do Provedor apresentam módulos com funcionalidades distintas e bem definidas. A camada de Orquestração contém três módulos:o de serviços, o de abstração de recursos e o de recursos físicos. A camada Controlador apresenta oito módulos distintos que proveem os recursos computacionais ao sistema. E por último tem-se a camada de Dependabilidade que apresenta dois módulos. O primeiro é o de atributos, que contém os dois principais atributos da dependabilidade. O segundo módulo é o de gerenciamento, que contém os componentes de monitoramento e controle. São descritos a seguir cada camada e os módulos do Provedor.

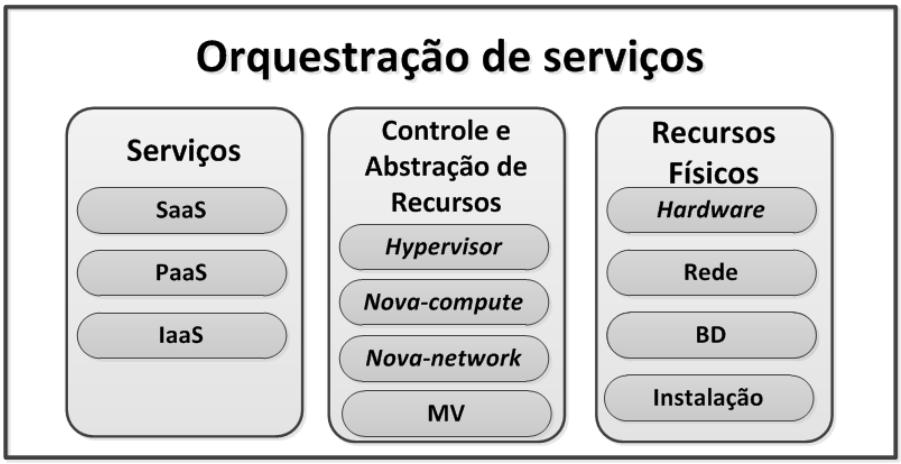
4.2.3.1 Camada de Orquestração

A camada de Orquestração de serviços possui módulos, cujo agrupamento é responsável pela entrega dos serviços do Provedor ao Consumidor. O primeiro módulo é o de

Serviços, que apresenta a forma pela qual os serviços são oferecidos aos Consumidores mediante modelos de serviços de Computação em Nuvem. Os serviços são classificados em três modelos [Liu et al. 2011]: SaaS-*Software* como Serviço, PaaS-Plataforma como Serviço e IaaS-Infraestrutura como Serviço. Como já citado anteriormente são as interfaces de acesso de cada um dos três modelos de serviços. Essas interfaces de acesso são fornecidas por este módulo.

O módulo de Abstração de Recursos contém os componentes do sistema que os Provedores utilizam para fornecer e gerenciar o acesso aos recursos de computação física por meio da abstração de *software*. Exemplos de componentes de captação de recursos são *hypervisors*, máquinas virtuais, armazenamento de dados virtuais e outras captações de recursos de computação. A abstração de recursos é realizada pelos módulos *Nova-compute, Nova-network* e pelo *hypervisor*. O *Nova-compute* e o *Nova-network* são componentes do *OpenStack*. A abstração de recursos é necessária para garantir o uso eficiente, seguro e confiável dos recursos físicos subjacentes. A Figura 4.6 ilustra os componentes de cada módulo da camada de Orquestração de serviços.

Figura 4.6: Componentes da camada Orquestração de Serviços



Fonte: Autora

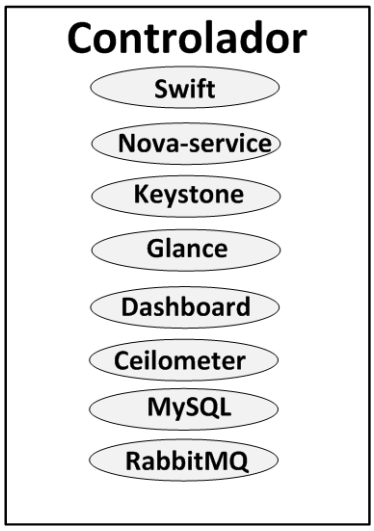
O último módulo da camada de Orquestração é o de Recursos Físicos; aqui se encontram todos os recursos de *hardware*, por exemplo, CPU, memória, componentes de redes, como roteadores, *firewalls* e comutadores, componentes de armazenamento, como discos rígidos e outros elementos físicos da infraestrutura. Os recursos de instalações, tais como o aquecimento, ventilação, bem como outros aspectos da planta física também fazem parte deste módulo.

4.2.3.2 Camada Controlador

A camada Controlador é a camada do meio do ator Provedor. Nela se encontram os componentes responsáveis por oferecer os recursos computacionais aos usuários do

Provedor de Computação em Nuvem. Esses componentes são chamados de módulos na ArqDep. Cada módulo é responsável por oferecer um serviço. A Figura 4.7 mostra os componentes do Controlador, cada componente tem a sua funcionalidade no sistema.

Figura 4.7: Componentes do Controlador



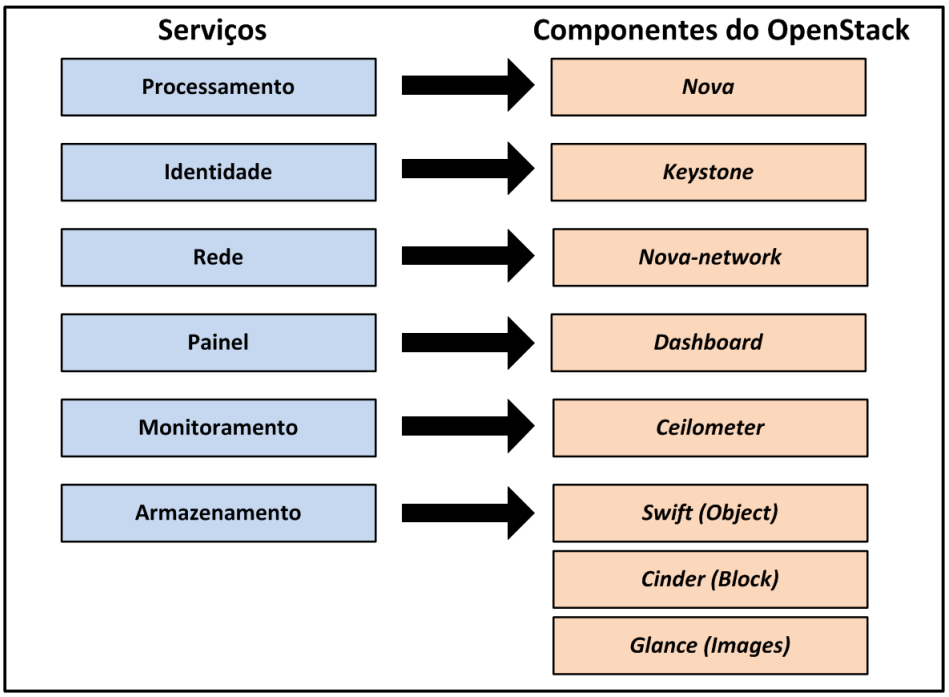
Fonte: Autora

O *Swift* fornece o serviço de armazenamento de objetos; o *Nova-service* engloba diversos componentes e é responsável por, basicamente, fornecer o serviço de criação de instâncias sob demanda; o *Keystone* tem a função de gerenciar os usuários da infraestrutura; o *Glance* fornece um repositório de imagens para a criação das instâncias virtuais; o *Dashboard* tem como função apresentar uma interface gráfica para a administração dos serviços; o *Ceilometer* é responsável pelo monitoramento do sistema, coletando informações dos recursos utilizados e fornecendo informações importantes em relação a desempenho, escalabilidade e estatísticas do sistema; o banco de dados utilizado pela infraestrutura e o responsável pela troca de mensagens.

A Figura 4.8 apresenta as funcionalidades, ou seja, os serviços prestados por cada um dos componentes do *OpenStack* utilizados na arquitetura proposta. Todo o fluxo de execução da infraestrutura é realizado nesta camada. Esses componentes, juntos, oferecem os serviços de Computação em Nuvem aos Consumidores.

O banco de dados utilizado na arquitetura é o MySQL, que utiliza a linguagem *Structure Query Language (SQL)*, que é uma das linguagens mais populares para inserir, acessar e gerenciar conteúdos armazenados num banco de dados.

Figura 4.8: Funcionalidades dos componentes do *OpenStack*



Fonte: Autora

Ele foi o escolhido por ser o padrão do *OpenStack* e por ser um banco de dados de código aberto. E as mensagens são realizadas com o *RabbitMQ* que foi construído utilizando a linguagem *Erlang* e implementa o *Advanced Message Queuing Protocol(AMQP)*. O AMQP dá suporte a vários tipos de comunicação, como *point-to-point* e *Publish-Subscribe*, esse último é o utilizado nesta arquitetura. Esses componentes atendem de forma eficaz às demandas da arquitetura.

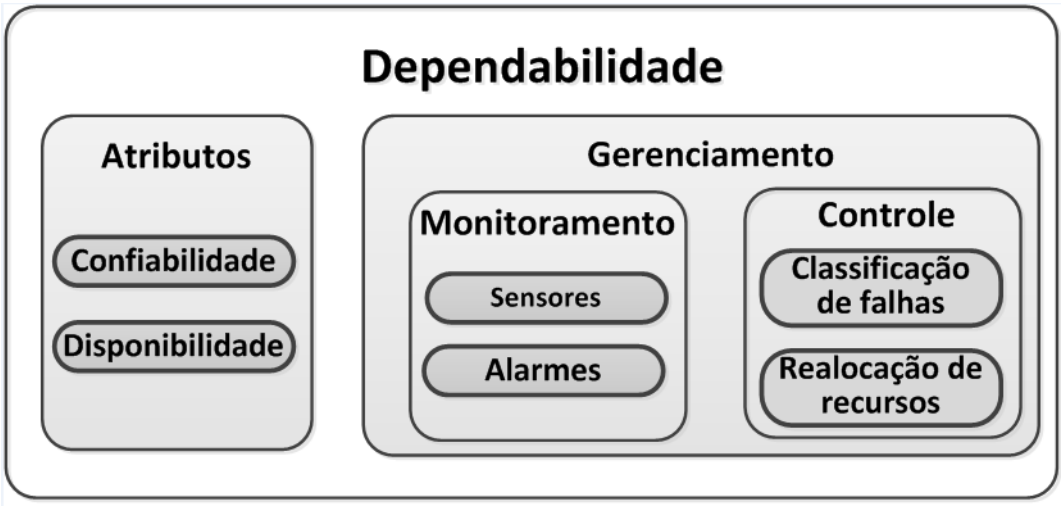
4.2.3.3 Camada de Dependabilidade

A última camada do Provedor da nuvem é a de Dependabilidade. Nessa camada há dois módulos: o de atributos e o de gerenciamento. No módulo de atributos têm-se os dois principais atributos da dependabilidade, que são utilizados como base nesta Dissertação de Mestrado, a disponibilidade e a confiabilidade. O segundo módulo é o de gerenciamento.

O módulo gerenciamento se divide em duas partes: o monitoramento e o controle. O monitoramento é responsável por coletar dados da infraestrutura, utilizando o *Ceilometer*. Como componentes do monitoramento têm-se os sensores e alarmes que podem ser acoplados a infraestrutura. O controle é responsável por realizar a classificação das falhas apresentadas na infraestrutura e a realocação de recursos. Para realizar essa classificação, é utilizado o modelo de classificação de falhas proposto nesta Dissertação de Mestrado. O controle também tem a função de realocar recursos no sistema. A Figura 4.9 ilustra os componentes de cada módulo da camada Dependabilidade.

Essa camada é a principal proposta desta Dissertação de Mestrado. A partir das

Figura 4.9: Componentes da camada Dependabilidade de serviços



Fonte: Autora

métricas coletadas e analisadas nessa camada, os administradores da infraestrutura podem tomar decisões que aperfeiçoem a prestação de serviços aos Consumidores. E na visão dos Consumidores, eles terão métricas para avaliar se os serviços prestados pelos Provedores estão de acordo com o que foi definido e assinado no contrato de SLA. De acordo com as métricas coletadas, o sistema pode tomar a decisão de realocar recursos entre as instâncias.

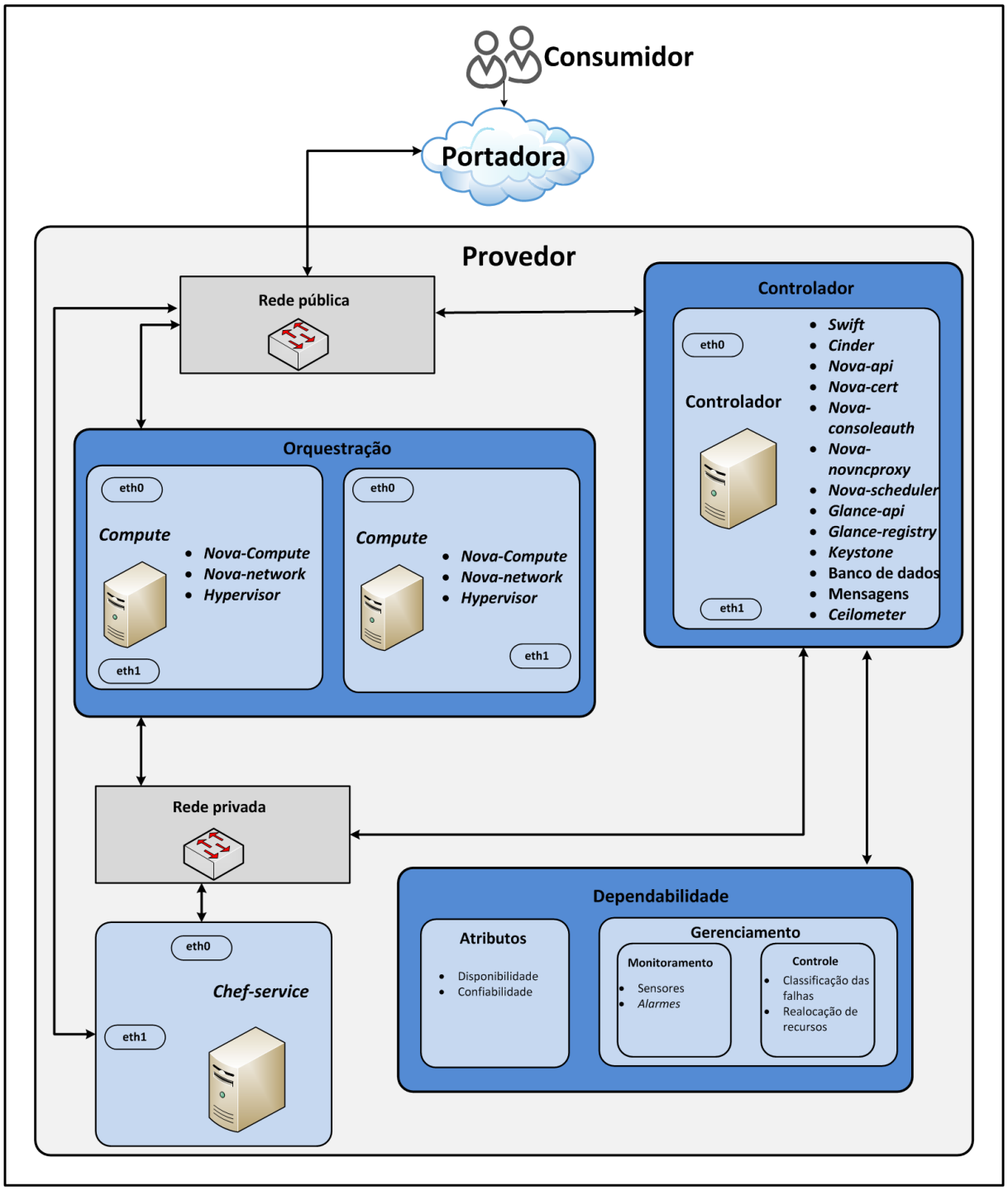
4.3 Implementação da ArqDep

Um dos objetivos específicos desta Dissertação de Mestrado é prover uma infraestrutura de Computação em Nuvem, com o modelo de implantação em nuvem privada. Como já mencionado anteriormente, a ArqDep é baseada na arquitetura de referência do NIST [Liu et al. 2011], incluindo algumas modificações e utilizando o gerenciador de nuvem *OpenStack* para realizar a implementação. A Figura 4.10 ilustra a topologia utilizada para a implementação da nuvem privada utilizando o *OpenStack*. Nesta seção é descrita a implementação da ArqDep, que é utilizada para a prova de conceito desta Dissertação de Mestrado.

Na Figura 4.10 pode-se destacar a presença dos três atores principais da ArqDep: Consumidor, Provedor e Portadora. No Provedor têm-se as camadas de Orquestração, Controlador e a Dependabilidade. A topologia da rede contém um nó *Chef-service* utilizado para configurar os servidores; esta máquina é opcional, optou-se em utilizar para automatizar o processo de configuração caso tenha necessidade de expandir a infraestrutura.

Na camada Controlador, tem-se o nó Controlador que contém os componentes do *OpenStack* que fornece os recursos computacionais para os usuários da infraestrutura. Na camada de Orquestração há dois nós *Compute* que são responsáveis pela execução e

Figura 4.10: Topologia da Nuvem Privada



Fonte: Autora

configuração das máquinas virtuais. E, por último, tem-se a camada de Dependabilidade, que é responsável pelo gerenciamento dos componentes da camada Controlador. Todas as máquinas da rede apresentam duas placas de redes e estão conectadas a rede pública e a rede privada.

Na topologia fica clara a divisão dos atores principais da ArqDep. Todos os nós da infraestrutura tem os seus respectivos componentes instalados. Esses componentes

são os responsáveis em prover os serviços de Computação em Nuvem. A camada de Dependabilidade monitora todos os recursos computacionais da infraestrutura, por meio dos módulos atributos e gerenciamento. No gerenciamento destaca-se os componentes de monitoramento e controle.

4.4 Infraestrutura como Serviço - IaaS

A seguir descreve-se como foi realizada a implementação da Infraestrutura como Serviço - IaaS em uma nuvem privada. Atualmente, muitas empresas têm deixado de alocar serviços em grandes Provedores de Computação em Nuvem para investir em sua própria infraestrutura de nuvem privada. Diversos motivos estão levando essas empresas em investirem dinheiro e tempo nessa tecnologia; o principal diz respeito à confidencialidade das informações. Recentemente, a mídia mundial relatou diversos casos de espionagem cibernética realizadas pelo governo americano [InfoEscola 2013] [Terra 2013]. Logo, investir recursos financeiros para reter todas as informações confidenciais da empresa tornou-se algo importante dentro das organizações.

Para realizar a implementação da infraestrutura de Computação em Nuvem, baseou-se em duas arquiteturas disponíveis na literatura. A primeira foi a arquitetura de referência do NIST [Liu et al. 2011] como mostra a Figura 4.1 e a segunda foi a arquitetura lógica do *OpenStack* [OpenStack 2013a]. A Figura 4.2 mostra a arquitetura lógica do *OpenStack*, como já descrito no capítulo anterior.

A implementação da infraestrutura de nuvem privada ocorreu no Laboratório de Sistemas Computacionais da Universidade Federal de Uberlândia. Para realizar a implementação da ArqDep, foi necessário montar um ambiente de simulações e testes, a configuração dos equipamentos utilizados neste ambiente são apresentados no Apêndice **??**.

1. Máquina Controlador com as seguintes especificações:

    - Processador Intel Core(TM) 2 Quad CPU Q2800 2,33 Ghz.
    - Memória 8GB.
    - Armazenamento 500GB.
    - Sistema Operacional Ubuntu 12.04.3 LTS.
    - Duas placas de rede.

2. Duas máquinas *Compute*, com o *hypervisor*, *Nova-compute* e *Nova-network* com as seguintes especificações:

    - Processador Intel Core(TM) 2 Quad CPU Q2800 2,33 GHz.
    - Memória 8GB.

- Armazenamento 500GB.
- Sistema Operacional Ubuntu 12.04.3 LTS.
- Duas placas de rede.

3. Uma máquina com o *Chef-service* com as seguintes especificações:

- Processador Intel Core i5-2500, 3,30 GHz, 6M cache, 4 cores/4 threads.
- Memória 8GB.
- Armazenamento 500GB.
- Sistema Operacional Ubuntu 12.04.3 LTS.
- Duas placas de rede.

4. Dois *switches* com 24 portas.

Esses foram os equipamentos utilizados para construção da infraestrutura de Computação em Nuvem. Em cada máquina da infraestrutura foram instalados componentes diversos para prover os recursos computacionais para os usuários da infraestrutura.

4.4.1 Máquina Controlador

O Controlador da nuvem executa os componentes que são responsáveis por prover os recursos computacionais sob demanda aos usuários. Os componentes instalados no Controlador foram: o *Swift*, o *Nova-service*, o *Glance*, *Cinder* o *Keystone*, o banco de dados, o servidor de mensagens e o *Ceilometer*.

Os serviços do *OpenStack* necessitam de um banco de dados para armazenar suas informações. O banco de dados instalado na infraestrutura foi o *MySQL*. No nó Controlador são instalados os pacotes cliente e servidor do MySQL e a biblioteca *Python*. Nos demais nós são instalados o cliente MySQL e a biblioteca *Python MySQL*. Algumas modificações devem ser feitas em alguns arquivos. Os comandos para a instalação estão no Apêndice A.

No Controlador é instalado o servidor de filas de mensagens. Nessa infraestrutura é utilizado o *RabbitMQ*; outros tipos de servidores de mensagens estão disponíveis, mas a escolha do *RabbitMQ* se deu por atender de forma eficaz as demandas da ArqDep. Os comandos utilizados para a instalação do *RabbitMQ* estão no Apêndice B. Após a instalação do *MySQL* e do *RabbitMQ* inicia-se a instalação dos componentes *OpenStack* que prover os serviços na nuvem.

O primeiro componente do *OpenStack* instalado no Controlador é o *Identity (Keystone)*. Tem como função gerenciar os usuários da infraestrutura, dar permissões e criar um catálogo dos serviços disponíveis. Os comandos utilizados para a instalação do *Keystone* estão no Apêndice C. Para compreender o funcionamento do *Keystone* é necessário entender os seguintes conceitos [OpenStack 2013c]:

- **Usuário:** É uma representação digital de uma pessoa, sistema ou serviço que utiliza os serviços da nuvem *OpenStack*. O usuário tem o seu *login* e são atribuídos *tokens* quando algum serviço é requisitado. Utilizando o *token* os usuários têm acesso aos recursos solicitados.
- **Credenciais:** São dados conhecidos por um usuário único para confirmar quem realmente é. Exemplos são o nome e a senha do usuário, nome do usuário e a chave da API ou um *token* de autenticação fornecido pelo *Keystone*.
- **Autenticação:** É a confirmação da identidade de um usuário. O *Keystone* confirma uma requisição de entrada, validando um conjunto de credenciais fornecidas pelo usuário. No primeiro momento, é solicitado o nome do usuário e a senha ou um usuário e a chave da API. Como resposta a solicitação, o *Keystone* emite um *token* de autenticação para o usuário.
- ***Token*:** É um texto usado para acessar os recursos. Cada *token* tem o seu alcance e são definidos quais os recursos são acessíveis com ele. Ele é válido por um período limitado de tempo e pode ser alterado a qualquer momento.
- ***Tenant*:** Um contêiner utilizado para agrupar ou isolar recursos, e ou objetos de identificação. Dependendo do operador de serviço, o *tenant* pode mapear para um cliente, conta, organização ou projeto.
- **Serviços:** São os serviços oferecidos pelos componentes do *OpenStack*.
- ***Endpoint*:** Um endereço acessível pela rede, geralmente descrito por *Uniform Resource Locator (URL)*, em que os usuários acessam os serviços.
- **Função:** Uma personalidade que o usuário assume permitindo executar um conjunto específico de operações. A função inclui um conjunto de direitos e privilégios. Um usuário assumindo essa função herda esses direitos e privilégios.

O *Glance* tem como função disponibilizar um catálogo e um repositório de imagens de discos virtuais, os quais são utilizados pelas máquinas virtuais executadas nas nuvens. Os comandos utilizados para a instalação do *Glance* estão no Apêndice D.

Os componentes do *Nova-service* são os principais componentes da infraestrutura, responsáveis por realizar o provisionamento dos recursos. O *Nova-service* interage com o *Keystone* para realizar as autenticações dos usuários, com o *Glance* para a utilização das imagens. Para criar uma instância, deve-se primeiro realizar o *download* das imagens. Ao criar uma instância, a imagem que será utilizada deve estar salva no *Glance*.

O acesso às imagens pode ser limitado por projetos ou usuários, por exemplo, o número de instâncias que podem ser criadas pode ser limitado. O *Nova-service* também interage com o *Dashboard* com as interfaces de usuário e administrativa. Os comandos utilizados para a instalação dos componentes do *Nova* estão no Apêndice E. O *Nova-service* é

composto de diversos subcomponentes. No Controlador foram instalados os seguintes componentes do *Nova-service*:

- *Nova-api:* Fornece uma interface para o mundo exterior interagir com a infraestrutura da nuvem. Oferece suporte a API *Nova* do *OpenStack*, a API da *Amazon EC2* e uma API *Admin* especial para usuários privilegiados para executar ações administrativas. Além disso, o *Nova-api* é responsável pela inicialização das instâncias.
- *Nova-cert:* Gerencia os certificados X.509.
- *Nova-consoleauth:* Fornece o serviço de autenticação para o *nova-console*. As solicitações dos usuários para acesso ao console são fornecidas com base nos *tokens* de autorização gerados pelo *Keystone*, permitindo o acesso a consoles virtuais por meio de um navegador para usuários autorizados.
- *Nova-novncproxy:* Fornece um *proxy* para acessar instâncias em execução por meio de uma conexão VNC.
- *Nova-scheduler:* Tem a função de registrar uma solicitação de uma instância de máquina virtual, a partir da fila e determina em qual nó do servidor *Compute* ela será executada.

O componente do *OpenStack Cinder* também é instalado no Controlador. O *Cinder* tem como função o fornecimento de armazenamento em blocos que pode ser montado como volume pelas instâncias. Ele gerencia a criação, anexação e liberação dos dispositivos de bloco para os servidores. Os volumes de armazenamento de bloco são integrados com o *Nova* e com o *Dashboard*. Os comandos utilizados para a instalação dos componentes do *Swift* são apresentados no Apêndice G. O *Cinder* apresenta os seguintes componentes:

- *Cinder-API:* Aceita as requisições e encaminha para o *Cinder-Volume*.
- *Cinder-Volume:* Funciona de acordo com as requisições escrevendo ou lendo na base de arquivos do *Cinder*.
- *Cinder-scheduler:* Escolhe o bloco mais adequado no provedor de armazenamento no qual deseja criar o volume.
- Fila de mensagens: Distribui as informações entre os Blocos de Armazenamento.

Outro componente do *OpenStack* instalado no Controlador é o *Swift*. Ele é um serviço equivalente com o *S3* da *Amazon*, cuja função é implementar um repositório de armazenamento de objetos eventualmente consistente. É totalmente escalável tanto em termos de tamanho, como também em capacidade. Os componentes do *Swift* instalados no Controlador foram o *Swift Proxy Server*, o *Swift Object Server*, o *Swift Container Server* e o *Swift Account Server*. Os comandos utilizados para a instalação dos componentes do *Swift* são apresentados no Apêndice F.

Por último deve ser feita a instalação do componente *Ceilometer*. Este componente é o responsável pelo monitoramento de toda a infraestrutura. Diversas métricas são obtidas utilizando esse componente. As métricas coletas ajudam os administradores da infraestrutura na tomada de decisões e aos usuários o acompanhamento dos recursos disponíveis pela infraestrutura. O *Ceilometer* contém os seguintes componentes: *Compute agent*, *Central agent*, *Collector*, *Data store* e o *Server API*. Todos esses componentes, exceto o *Compute agent* foram instalados no Controlador. O componente *Compute agent* é instalado no *Compute*. Os comandos utilizados para a instalação dos componentes do *Ceilometer* são apresentados no Apêndice H.

4.4.2 Máquinas *Compute*

As máquinas *Compute* têm a função de executar o *hypervisor* e alguns componentes do *Nova-service*. Os componentes do *Nova-service* instalados na máquina *Compute* são o *Nova-Compute* que é o componente principal, chamado de núcleo do *Nova* e o *Nova-network* que é responsável pelas configurações de redes da infraestrutura. Os comandos utilizados para a instalação dos componentes do *Nova* estão no Apêndice E.

O *Nova-Compute* basicamente é responsável pelo ciclo de vida das máquinas virtuais. Ele recebe requisições de gerenciamento mediante as filas de mensagens e executam as operações que são necessárias. Na infraestrutura desta Dissertação de Mestrado há dois nós com *Nova-Compute*. Em uma infraestrutura de Computação em Nuvem podem existir vários *Nova-Compute* instalados. Uma instância pode ser criada por qualquer *Nova-Compute* disponível na infraestrutura, a escolha ficará por conta do algoritmo de escalonamento utilizado.

O *Nova-network* funciona como um gerenciador da rede. Tem como função a configuração da rede das máquinas. Ele é responsável pela alocação de endereços de IPs, configurações de VLANs e implementação de grupos de segurança. Na versão *Havana* do *OpenStack* utiliza-se o componente *Neutron* para as configurações de rede, nesta Dissertação de Mestrado não foi utilizado; optou-se pelo *Nova-network* pois até o momento do desenvolvimento dessa infraestrutura o *Neutron* não permitia a configuração de *Multi-host*, que é a utilizada nesta Dissertação de Mestrado.

4.4.3 *Hypervisor QEMU*

O *Hypervisor Quick EMUlator (QEMU)* é um *software* de virtualização livre escrito por Fabrice Bellard. Ela tem a função de emular um processador, além de permitir a emulação de todos os subsistemas necessários, como *hardware* de interligação de redes e de vídeo. O *QEMU* permite a virtualização completa de um sistema dentro de outro. Ele apresenta diversas características semelhantes aos *hypervisors* existentes no mercado, mas uma diferenciação apresentada por este *hypervisor* é o acelerador, que aumenta a

velocidade da emulação em arquiteturas x86 [QEMU 2013].

O *QEMU* possui dois modos de operação: a emulação em modo de usuário e a emulação em modo de sistema. A primeira possibilita que um processo construído para uma CPU possa ser executado em outra. E a segunda é a emulação em modo de sistema que permite a emulação de um sistema integral, incluindo o processador e periféricos variados [DeveloperWorks 2013].

O *QEMU* apresenta um conversor dinâmico que permite a conversão em tempo de execução de instruções para uma CPU de destino para a CPU *host* para fornecer emulação. Isso não é simples de se fazer, em muitas vezes é realizado por força bruta, mapeando instruções de uma CPU para outra, em alguns casos, pode requerer múltiplas instruções ou mudanças no comportamento com base nas arquiteturas sendo convertidas [DeveloperWorks 2013].

4.4.4 Máquina *Chef-service*

A máquina *Chef-service* contém o *software Chef*. O *Chef* é uma ferramenta *open source* para integração de sistemas e gerenciamento de configuração. O *Chef* é usado para descrever como os servidores devem ser configurados, incluindo pacotes que devem ser instalados, serviços que precisam estar rodando e arquivos que precisam ser criados ou editados. O *Chef* verifica se cada recurso está devidamente configurado e só efetua medidas corretivas se forem necessárias, garantindo que os servidores estejam sempre funcionado exatamente como devem estar.

O *Chef-server* automatiza a infraestrutura, mantendo os servidores padronizados com as configurações atuais. Ele é um *framework* de código aberto de integração de sistemas criado especialmente para automatizar tarefas nas nuvens, permitindo por meio dos *cookbooks* que os administradores da infraestrutura facilmente provisionem, gerenciem e escalem seus servidores [Opscode 2013].

Quando um novo servidor é adicionado, basta informar ao *Chef* quais as funções ele deve desempenhar na arquitetura e ele fará todas as configurações necessárias e depois vai garantir que continuem os serviços da forma correta. Essa ferramenta é utilizada para deixar o nosso ambiente de produção automatizado, não sendo mais necessário realizar as configurações em todos os servidores adicionados na infraestrutura.

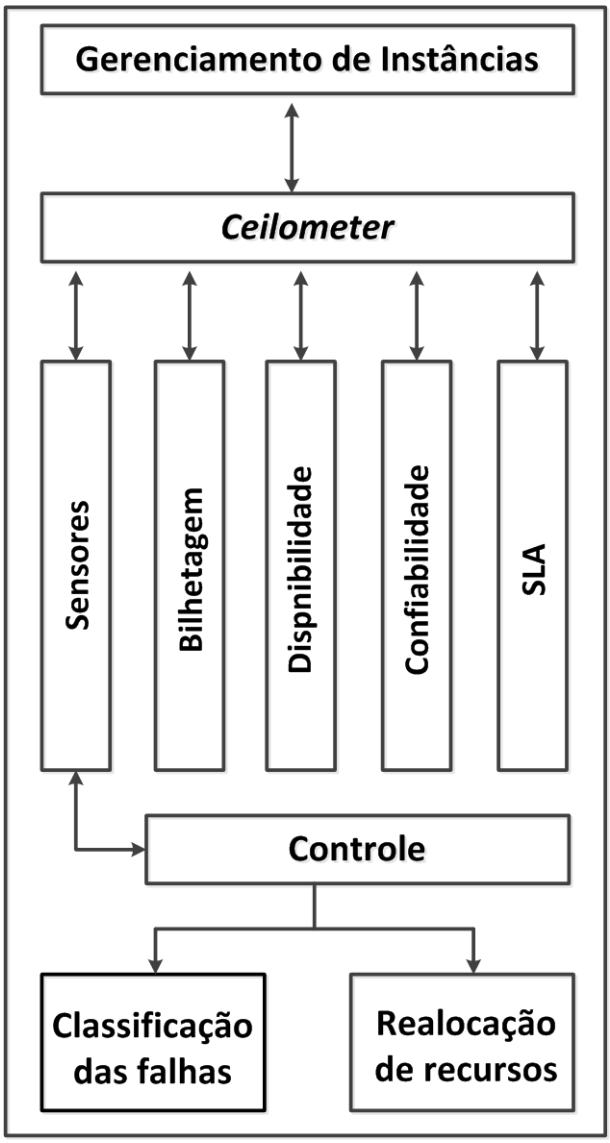
4.5 Módulo de gerenciamento

O módulo de gerenciamento de infraestrutura de Computação em Nuvem foi projetado nesta Dissertação de Mestrado e tem duas funções básicas: o de monitoramento e o de controle. De uma forma geral, o objetivo do módulo de gerenciamento é monitorar e controlar os componentes responsáveis pelos serviços de Computação em Nuvem. Logo,

as instâncias virtuais do sistema são monitoradas durante toda a sua vida, por meio do componente do *OpenStack Ceilometer*.

Diversos sensores com objetivos distintos podem ser acoplados no módulo de gerenciamento, por exemplo, sensores que analisam os *logs* gerados pelo sistema. Esse módulo de gerenciamento gera relatórios de bilhetagem, disponibilidade, confiabilidade e SLA. A estrutura do sistema de gerenciamento proposto nesta Dissertação de Mestrado é ilustrada pela Figura 4.11.

Figura 4.11: Módulo de gerenciamento de infraestruturas de Computação em Nuvem



Fonte: Autora

No topo do sistema tem-se o Gerenciamento de instâncias. O objetivo desta camada é gerenciar todas as instâncias criadas na infraestrutura. Todos os recursos computacionais das instâncias são monitorados pelo componente da camada inferior, o *Ceilometer*, que funciona como o *middleware* do sistema de gerenciamento. Na camada Ceilometer podem ser acoplados diversos componentes, como sensores, geradores de relatórios de bilheta-

gem, relatórios de disponibilidade, relatórios de confiabilidade e relatórios referentes ao SLA. Outros componentes podem ser acoplados no sistema, dependendo das necessidades dos usuários do módulo de gerenciamento. No módulo de gerenciamento proposto os componentes acoplados são:

- **Sensores:** Vários sensores podem ser acoplados na infraestrutura para monitorar algum recurso específico. Alarmes podem ser configurados, para avisar aos administradores algo que saia da normalidade no funcionamento do sistema. É proposto neste módulo de gerenciamento um sensor de análise de falhas. Este sensor analisa os *logs* das instâncias de máquinas virtuais e envia as informações das falhas encontradas para a camada de Controle.
- **Bilhetagem:** Esse componente é responsável por analisar os dados de cobrança coletados pelo *Ceilometer* e elaborar os relatórios de cobrança aos usuários da infraestrutura.
- **Disponibilidade:** O componente disponibilidade acoplado no *Ceilometer* gera relatórios sobre a disponibilidade das instâncias de máquinas virtuais.
- **Confiabilidade:** Do mesmo modo do componente anterior, relatórios de confiabilidade podem ser criados. As métricas utilizadas para o cálculo da confiabilidade são obtidas a partir das métricas dos recursos computacionais coletados pelo *Ceilometer*.
- **SLA:** Relatórios referentes ao SLA são elaborados em cima das métricas obtidas pelo *Ceilometer*. Estes relatórios podem colaborar com os usuários finais da infraestrutura, auxiliando na verificação se a prestação de serviço está de acordo com o contrato assinado no fechamento do negócio.

O sensor que captura as falhas ocorridas na infraestrutura de Computação em Nuvem envia as suas informações para a camada de Controle. A camada de Controle tem duas funções: a de classificação das falhas e a de realocação de recursos. A classificação das falhas utiliza um modelo de classificação, esse modelo se baseia no que ocasionou esse desvio correto do funcionamento do sistema. Este modelo de classificação é descrito na seção a seguir.

Uma das tarefas mais importantes para uma infraestrutura de Computação em Nuvem é a realocação de recursos. A realocação de recursos envolve a prerrogativa adequada entre os requisitos do serviço de um consumidor e as características de recursos do provedor de acordo com o SLA. A maioria dos mecanismos de realocação de recursos utiliza algoritmos dinâmicos, que visam aumentar o desempenho da infraestrutura evitando que recursos computacionais fiquem ociosos na infraestrutura. É criado uma lista de recursos disponíveis que podem ser realocados para outras instâncias da infraestrutura.

4.5.1 Classificação das falhas

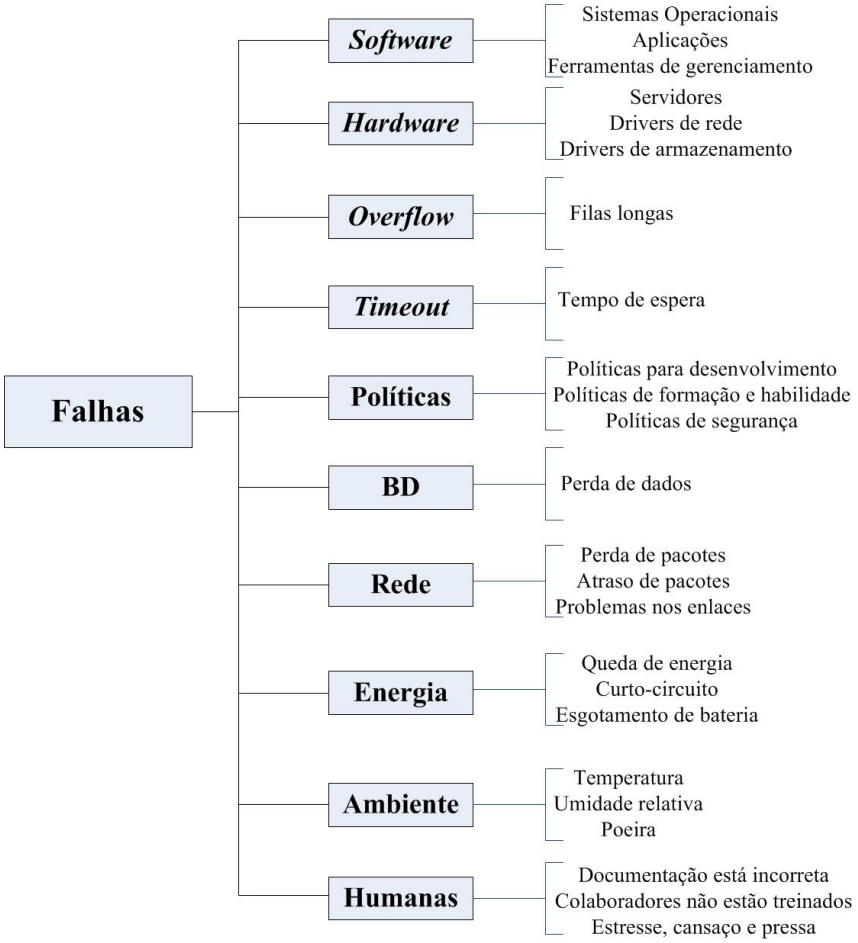
Analisar e modelar a confiabilidade da nuvem não é uma tarefa trivial, devido à complexidade em grande escala do sistema. A falha é um evento que ocorre quando a função realizada pelo sistema não está de acordo com a correta especificação que foi definida para a sua execução, essas definições são oriundas da Engenharia de *Software* [Musa et al. 1987]. Diversas falhas são detectadas no ambiente de Computação em Nuvem tais como: *Overflow*, *Timeout*, falha de rede, falha de *hardware*, falha de *software*, falha no banco de dados e outras. Elas falhas são descritas a seguir [Dai et al. 2009] [Bauer e Adams 2012]:

- ***Overflow:*** A fila de solicitação deve ter um limite máximo, de solicitações em espera. Isto é, quanto maior for a fila, maior será o tempo de espera. Caso contrário, novas solicitações têm de esperar por um tempo muito longo na fila, o que poderia tornar as falhas de limite de tempo muito mais dominantes. Portanto, quando a fila estiver cheia, e uma nova solicitação chegar, ela é simplesmente descartada e o usuário será incapaz de começar o serviço;
- ***Timeout:*** São as falhas de limite de tempo ocasionados por longos tempos de espera. Por exemplo, entre a comunicação de dois dispositivos A e B. Essas falhas ocorrem quando o tempo de resposta do dispositivo B é maior que a latência da comunicação entre os dispositivos A e B ou quando o tempo de resposta do dispositivo B é maior que o tempo limite do dispositivo A;
- **Falha de *hardware:*** Computadores e servidores podem apresentar falhas de hardware;
- **Falha de *software:*** As subtarefas são na verdade programas de *software* rodando em diferentes recursos computacionais, que podem conter falhas de *software*;
- **Falha no Banco de Dados:** O banco de dados armazena os dados necessários, também pode falhar, fazendo com que as subtarefas, durante a execução, não possam acessar os dados necessários;
- **Falha de Rede:** Quando subtarefas acessam dados remotos ou os canais de comunicação, elas podem ser quebradas fisicamente ou logicamente possibilitando a geração de falhas de rede;
- **Energia:** São as falhas decorrentes da parte elétrica do sistema. Esta categoria inclui os picos de sobretensão, a queda de energia e os curtos circuitos. Essas falhas são suscetíveis a sistemas com alimentação externa. Sistemas movidos a baterias são também vulneráveis a riscos, tais como desgaste natural da bateria e o esgotamento da carga;

- **Ambiente:** O ambiente deve ser adequado para implantação de todo o sistema, de forma que exista espaço físico suficiente para os engenheiros manter o sistema. O ambiente deve ser altamente protegido para evitar falhas deliberadas ou acidentais;
- **Falhas humanas:** Os seres humanos operam, usam e mantêm o sistema. As falhas humanas podem ocorrer a qualquer momento no sistema por muitas razões, por exemplo, pela falta de treinamento adequado para manutenção do sistema;
- **Políticas:** Para operar com sucesso um sistema complexo é necessário ter políticas de negócios, bem definidas, dentro das empresas. Uma falha no gerenciamento do negócio pode acarretar em grandes perdas financeiras.

A Figura 4.12 ilustra o modelo de classificação de falhas. Esta proposta é uma contribuição adicional nesta Dissertação de Mestrado, pois na literatura encontram-se a de Dai e Bauer [Dai et al. 2009] [Bauer e Adams 2012], mas não há uma classificação completa como esta. Primeiramente classificam-se as falhas mais frequentes encontradas na nuvem de acordo a literatura. Em seguida apresentam-se os meios que podem ocasionar uma falha no sistema.

Figura 4.12: Modelo de classificação de falhas



Fonte: Autora

Esse modelo de classificação de falhas é utilizado na ArqDep. As falhas apresentadas na infraestrutura podem ser analisadas e classificadas de acordo com o modelo de classificação proposto. Diversas outras falhas podem ser encontradas na infraestrutura, pois esse modelo ainda não contempla todas as falhas possíveis em um ambiente de Computação em Nuvem.

4.5.2 Realocação de recursos

Um dos maiores desafios encontrados pelos provedores de serviços de Computação em Nuvem é o provisionamento dos recursos computacionais, em ambientes heterogêneos, em que a infraestrutura apresenta uma grande variedade de máquinas. Cada máquina na infraestrutura pode apresentar características distintas, o que dificulta o provisionamento de recursos nestes ambientes.

A maioria das infraestruturas existentes de Computação em Nuvem utiliza métricas de quantidade de CPU e memória para realizar o escalonamento dos recursos computacionais entre as máquinas virtuais. Como os ambientes de Computação em Nuvem apresentam heterogeneidade, têm-se máquinas com diferentes quantidades de CPU com diferentes capacidade de processamento. Estas diferenças podem influenciar no desempenho das máquinas virtuais e consequentemente nas aplicações que estiverem encapsuladas nelas.

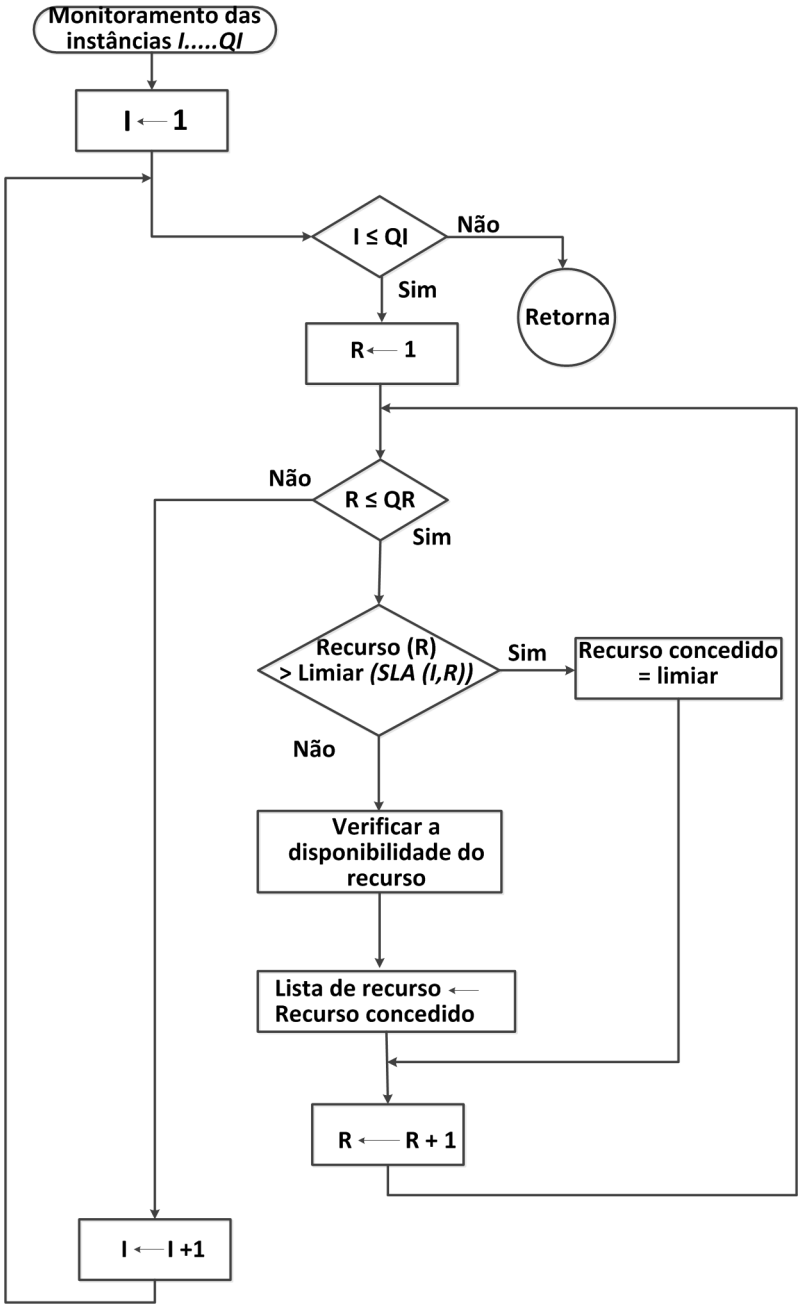
As infraestruturas devem atender as demandas dos consumidores de forma escalável e elástica, para atender as características da Computação em Nuvem. Os processos de alocação e realocação de recursos devem ser dinâmicos e transparentes aos consumidores. O provedor de Computação em Nuvem deve garantir que todos os requisitos fechados no SLA devam ser garantidos na infraestrutura.

Os requisitos a serem atendidos preconizam que, qualquer mecanismo utilizado para realocação de recursos da infraestrutura deve conhecer o estado de todos os elementos da infraestrutura. Logo, os mecanismos deve aplicar algoritmos para realocação de recursos, aproveitando os recursos virtuais e físicos da infraestrutura da melhor forma possível. Um mecanismo de realocação de recurso, pode ser visto como qualquer mecanismo que visa a garantir que os requisitos contratados pelos consumidores junto ao provedor sejam atendidos.

Os mecanismos de realocação de recursos têm como objetivo diminuir o custo operacional e aumentar o desempenho da infraestrutura. A utilização dos mecanismos de realocação de recursos é necessária para evitar o sub-provisionamento de recursos ou o excesso de provisionamento, garantindo, assim, o uso adequado dos recursos computacionais da infraestrutura de uma forma dinâmica.

A ArqDep utiliza um algoritmo de realocação de recursos para fornecer aos seus consumidores as características de escalabilidade e elasticidade da Computação em Nuvem. A Figura 4.13 apresenta o fluxograma para o escalonamento de recursos da infraestrutura.

Figura 4.13: Fluxograma para realocação de recursos



Fonte: Autora

Todas as instâncias da infraestrutura são monitoradas durante todo o tempo. São monitoradas as instâncias de $I.....QI$ e os recursos de $R.....QR$. Onde a variável QI representa a quantidade de instâncias e a variável QR representa a quantidade de recursos monitorados pela infraestrutura. Onde, a variável I indica a instância que está sendo monitorada e a variável R indica o tipo de recurso que está sendo monitorado.

O algoritmo representado pelo fluxograma apresentado pela Figura 4.13 monitora todas as instâncias e todos os recursos da infraestrutura. Quando um determinado recurso é solicitado ele analisa se o recurso solicitado é maior do que o acordado no SLA usando

Algoritmo 1: Algoritmo realocação de recursos

```
Entrada: Lista de Instâncias I
R ← Lista de recursos;
início
    enquanto i ← 1 até QI faça
        enquanto j ← 1 até QR faça
            se RecursoSolicitado(I_i, R_j) > Limiar(SLA(I_i, R_j)) então
                Recurso Concedido (I_i, R_j, Limiar(SLA(I_i, R_j)));
            senão
                Verificar a disponibilidade do recurso (I_i, R_j, ΔR_j);
                Lista de Disponibilidade ← Recurso Concedido
                (I_i, R_j, Disponibiliza(I_i, R_j, ΔR_j));
            fim
        fim
    fim
fim
```

a função *Limiar(SLA(I_i, R_j))*. Se o recurso solicitado estiver acima do *limiar*, o recurso concedido será igual ao *Limiar(SLA(I_i, R_j))*. Caso contrário, se o recurso solicitado estiver inferior ao acordado no SLA, o algoritmo verifica a disponibilidade do recurso ocioso por meio da função $Verificar\_Disponibilidade(I_i, R_j)$ e inclui o recurso ocioso na lista de disponibilidade: $Lista\_de\_Disponibilidade \longleftarrow Recurso\_Concedido$. A lista de disponibilidade contém todos os recursos ociosos disponíveis na infraestrutura que podem ser utilizados por qualquer instância de máquina virtual.

O algoritmo 1 apresenta o escalonamento de recursos realizados na infraestrutura. A utilização deste algoritmo dá aos consumidores dos serviços de Computação em Nuvem da infraestrutura uma garantia em relação a utilização dos recursos acordados no SLA. Já ao provedor dos serviços tem a otimização da infraestrutura não tendo recursos ociosos, assim têm-se uma diminuição do custo operacional da infraestrutura. Com isso, o provedor pode oferecer os seus serviços a um número maior de consumidores, pois a probabilidade de todos os usuários da infraestrutura utiliza-lá ao mesmo tempo é menor. Assim, é possível disponibilizar o sistema a um número maior de consumidores do que a infraestrutura realmente comporta.

O algoritmo de realocação de recurso é executado na ArqDep em períodos pré-determinados. A cada período uma nova verificação é realizada e a lista de disponibilidade é atualizada. São estabelecidos limites sobre a disponibilidade dos recursos, seguindo o que foi acordado no SLA. Existem diversas formas de construir este limiar de recurso, levando em consideração o SLA. Pode-se limitar a utilização de recursos, de acordo com o SLA por:

- Limite máximo fixo. Que é o utilizado no algoritmo descrito acima.
- Limite mínimo fixo.
- Limites flexíveis, o provedor garante recursos infinitos aos consumidores.

- Intervalo de limites, no SLA descreve um valor mínimo e um máximo de recursos disponíveis. Os recursos devem variar neste intervalo.
- Limite fixo.

Através do monitoramento contínuo da infraestrutura, têm-se as métricas de utilização da CPU e da memória. O algoritmo verifica as máquinas virtuais da infraestrutura e se uma máquina virtual atingir o limite pré-estabelecido de utilização da CPU e da memória, ele realoca os recursos computacionais das máquinas virtuais disponíveis para atender da melhor forma as necessidades do consumidor.

4.6 Avaliação da Dependabilidade

Para avaliar a dependabilidade de uma infraestrutura de Computação em nuvem, é preciso entender o impacto dos serviços da nuvem no desempenho da aplicação. Os recursos computacionais da nuvem, quanto ao comportamento das aplicações implantadas que utilizam esses recursos devem ser analisados. Os recursos de processamento, memória, largura de banda, latência, armazenamento e disponibilidade são monitoradas e analisadas durante toda a vida útil do sistema.

Ao realizar essa análise tem-se então como identificar os parâmetros ideais de configuração dos componentes das aplicações, do modo de implantação das aplicações e do provisionamento adequado dos recursos da nuvem as aplicações. Baseando-se no modelo de classificação de falhas proposta na seção 4.5.1, foi construída uma Árvore de falhas como visto na Figura 4.14, que é um método para análise da dependabilidade.

4.6.1 Árvore de Falhas

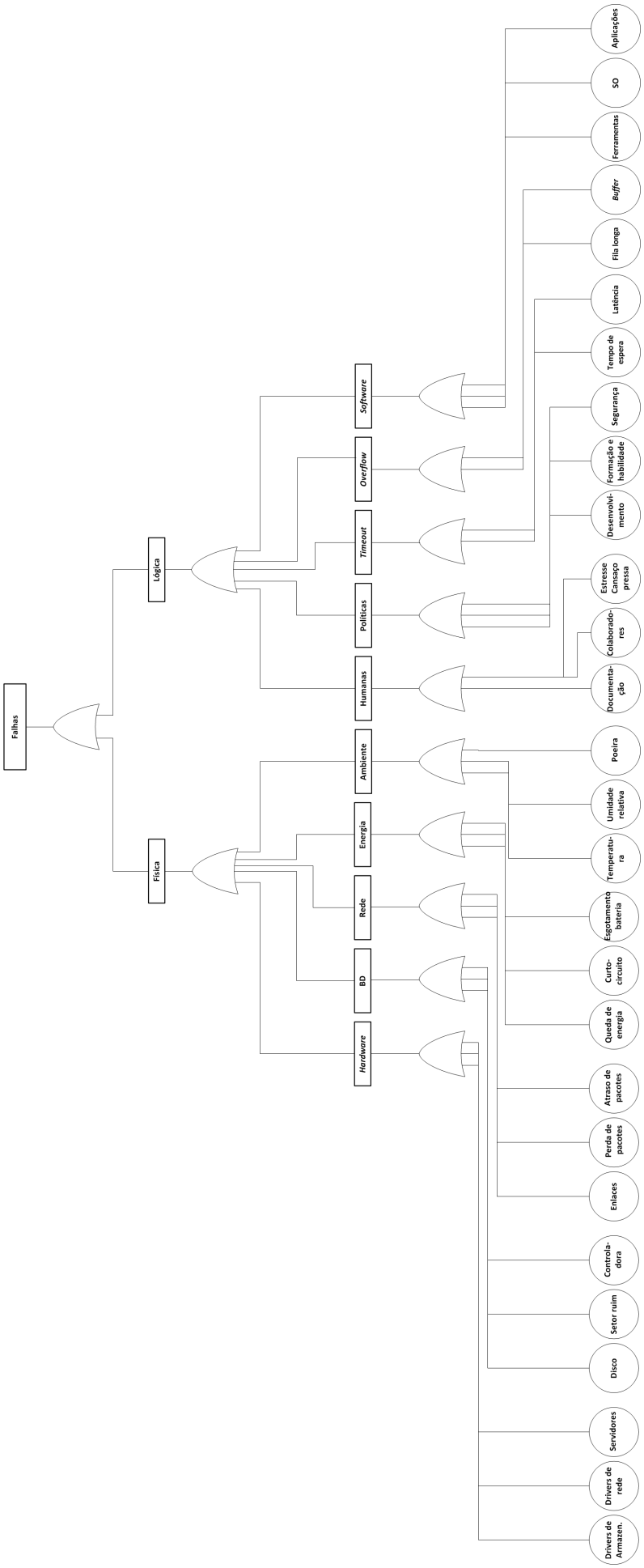
O método de análise Árvore de Falhas pode ser chamado do tipo *Top-Down*, pois a sua análise se inicia com um evento indesejável no topo da árvore e só após são determinadas todas as formas que levam este evento do topo a ocorrer. Diversas métricas de dependabilidade podem ser obtidas através da Árvore de Falha. A Figura 4.14 ilustra a Árvore de falhas do sistema.

A falha do sistema é representada pelo evento no *Topo* da árvore, logo a probabilidade deste evento ocorrer durante o período de tempo t é o complemento da confiabilidade $R(t)$. O cálculo da confiabilidade é realizado utilizando os conceitos de probabilidade dos eventos das portas lógicas, por exemplo *AND* e *OR*.

Assume-se que o sistema a ser analisado é composto de n componentes independentes e cada componente i tem a sua própria função de confiabilidade $R_i(t)$. Optou-se por construir somente uma árvore de falha geral para a arquitetura. Todos os componentes que podem levar ao sistema a falhar são detalhados na árvore de falha. No topo da árvore

tem a Falha no Sistema e todos os componentes que podem levar a falha do sistema são descritos na árvore.

Figura 4.14: Árvore de falhas do sistema



Fonte: Autor

4.7 Métricas Coletadas

O *Ceilometer* tem a função de monitorar o sistema durante toda a sua vida útil. Ele captura diversas métricas das instâncias das máquinas virtuais de forma contínua. Na infraestrutura construída, foram criadas diversas instâncias e serão detalhas métricas de três instâncias criadas. As configurações das instâncias são definidas na Tabela 4.1.

Tabela 4.1: Instâncias criadas

Instância	Nome	Imagem	Tamanho	ID
Instância A	*Cirros2*	*cirros1*	*m1.tiny/512MB RAM/1 vCPU/1,0GB Disco*	b8c7048f-90b6-41d3-9165-ab0ff67a46fd
Instância B	*Fedora*	*fedora19*	*m1.small/2GB RAM/1 vCPU/20,0GB Disco*	6ae20270-6bf3-4af4-a47f-10f5df8fa30d
Instância C	*Ubuntu-image*	*ubuntu*	*m1.small/2GB RAM/1 vCPU/20,0GB Disco*	6da92167-e167-4cff-9bca-fe8caaa65afb

Algumas métricas que afetam a disponibilidade e a confiabilidade do sistema serão discutidas a seguir. O sistema de monitoramento é composto por diversas opções, que são selecionadas pelos usuários por meio do *Dashboard*. As métricas podem ser selecionados por:

- **Agrupamento**: por projeto ou instâncias individuais.
- **Valor**: O usuário pode escolher pelo valor médio, mínimo, máximo e soma das métricas coletadas.
- **Período**: os períodos são selecionados de acordo a necessidade do usuário. As seguintes opções estão disponíveis:
  - Último dia.
  - Última semana.
  - Últimos 15 dias.
  - Últimos 30 dias.
  - Último ano.
  - Outro, em que se pode escolher a data de inicio e a data final do monitoramento.

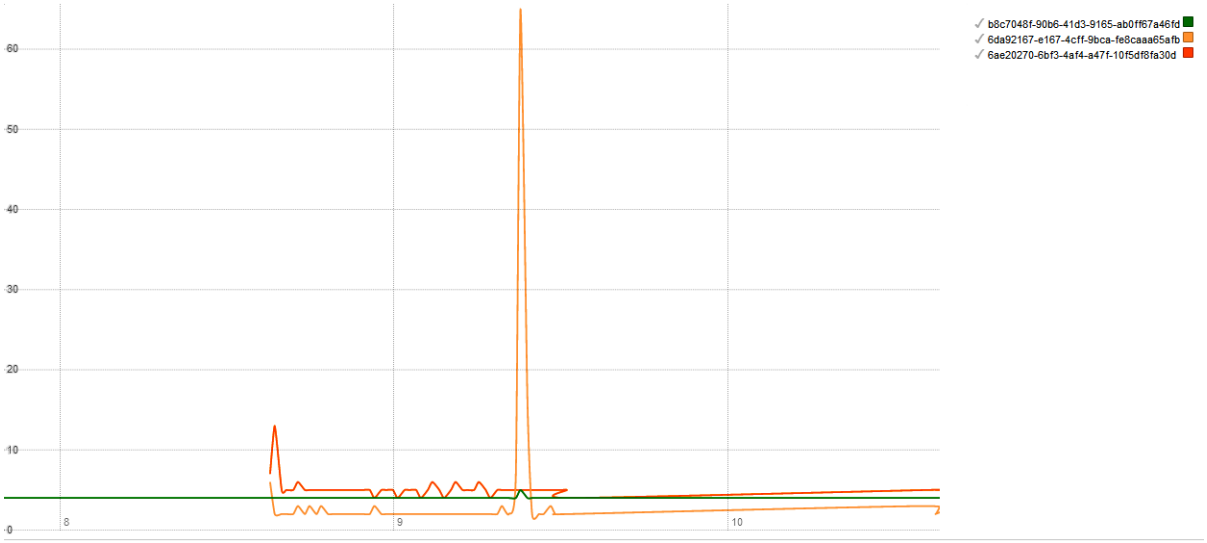
4.7.1 Discussões e resultados

Esta seção apresenta os resultados do monitoramento realizado na infraestrutura. O monitoramento pode ser realizado por meio do *Dashboard* que mostra de forma gráfica as métricas coletadas de todos os recursos disponíveis na infraestrutura ou por linha

de comando. Como citado anteriormente, o usuário do *Dashboard* pode refinar o seu monitoramento escolhendo as opções disponíveis para visualização das métricas.

A primeira métrica de monitoramento apresentada é a de utilização CPU. Esta medida mostra o tempo de utilização da CPU. A Figura 4.15 ilustra os dados de utilização da CPU na última semana de execução da infraestrutura, o agrupamento utilizando foi o por instâncias e o valor foi a média aritmética simples. Foram selecionadas três instâncias como mostra a Figura 4.15.

Figura 4.15: Monitoramento da utilização da CPU



Fonte: *OpenStack*

A instância A é representada pela reta de cor verde, a instância B é representada pela reta com a cor laranja escuro e a última instância C é representada pela reta de cor laranja claro. Esta métrica também pode ser monitorada por projeto, mudando somente a forma de selecionar os dados no *Dashboard*. O eixo X do gráfico ilustrado na Figura 4.15 apresenta o período de monitoramento, neste caso são os dias 8, 9 e 10 da última semana. Já o eixo Y apresenta a porcentagem de utilização da CPU.

Pelo gráfico apresentado na Figura 4.15 nota-se que a instância A não sofreu grande alteração na utilização da CPU. A instância B oscilou a sua utilização, mas de uma forma suave. Já a instância C entre os dias 9 e 10 apresentou um pico de consumo elevado da CPU. A utilização da CPU pode ser escalonada utilizando o Algoritmo 1. O limiar de cada recurso da infraestrutura é decidido levando em consideração o SLA.

Por meio de linha de comando pode-se listar as métricas que o *Ceilometer* disponibiliza utilizando o seguinte comando:

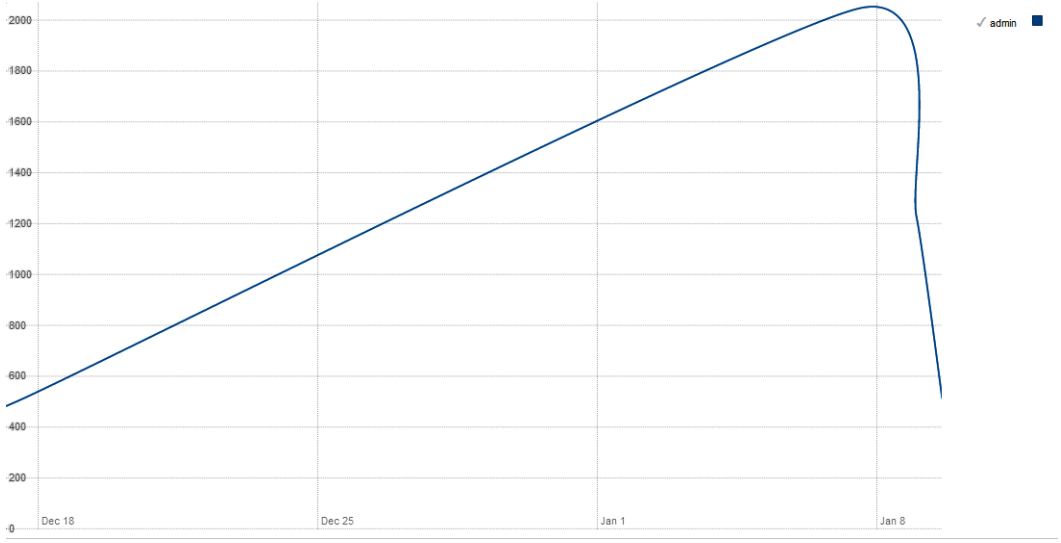
 # ceilometer –os -username [username] –os-password [password] –os-tenant-id [tenant-id] –os-auth-url http://localhost:5000/v2.0 meter-list

Para coletar uma métrica específica, por exemplo, utilização de CPU:

 # ceilometer –os -username [username] –os-password [password] –os-tenant-id [tenant-id] –os-auth-url http://localhost:5000/v2.0 sample-list –meter cpu_ util

O segundo recurso apresentado que pode ser monitorado pela infraestrutura é a memória. A Figura 4.7.1 ilustra a utilização da memória para o projeto *admin*. O sistema de monitoramento foi refinado para gerar a figura no agrupamento projeto, também pode realizar o agrupamento por instâncias. O período selecionado é dos últimos 30 dias e o valor a média da memória em MB.

Figura 4.16: Monitoramento da memória



Fonte: *OpenStack*

O gráfico apresentado na Figura mostra que houve um crescimento da utilização da memória da infraestrutura até a data de 8 de janeiro, após esta data a utilização da memória pelo projeto *admin* caiu. Neste projeto que foram criadas as instâncias de máquinas virtuais analisadas. A métrica de utilização de memória também pode ser coleta via linha do comando por meio do seguinte comando:

 # ceilometer –os -username [username] –os-password [password] –os-tenant-id [tenant-id] –os-auth-url http://localhost:5000/v2.0 sample-list –meter memory

A utilização do recurso memória pode ser controlado pelo Algoritmo 1. Este controle deve seguir as métricas acordadas no SLA assinado entre o provedor dos serviços de Computação em Nuvem e os usuários. Um limiar da utilização deste recurso é criado, de acordo com o SLA e os recursos em excesso ou livres podem ser realocados para outras instâncias

da infraestrutura. Assim, aumenta a eficiência da infraestrutura, a disponibilidade dos serviços e a sua confiabilidade.

Diversos outros recursos podem ser monitorados por meio desta infraestrutura. As tabelas 3.4, 3.5, 3.6, 3.7 e 3.8 mostram os recursos que podem ser monitorados por meio do *Ceilometer*. Este monitoramento e controle tem como objetivo aumentar o desempenho da infraestrutura. Oferecendo aos usuários um ambiente com disponibilidade e confiabilidade.

4.8 Conclusão

Neste capítulo foi descrita a ArqDep. Todos os componentes presentes na arquitetura foram detalhados, bem como as relações apresentadas entre os componentes. A ArqDep foi implementada utilizando o modelo de serviço IaaS em uma nuvem privada, para que os administradores da infraestrutura tenham maior controle sobre ela.

As principais contribuições desta dissertação foram apresentadas neste capítulo. Tem-se o módulo de gerenciamento que realiza o monitoramento e o controle da infraestrutura. Tal módulo foi responsável por realizar a classificação das possíveis falhas apresentadas na infraestrutura de acordo com o modelo de classificação de falhas proposto e por realizar a realocação de recursos.

Foi criado um algoritmo que tem como função realocar recursos ociosos da infraestrutura. Assim, a infraestrutura aumenta o seu desempenho e a sua eficiência. Uma avaliação de dependabilidade foi realizada utilizando os conceitos de árvore de falhas e essa avaliação baseou-se no modelo de classificação de falhas proposto neste capítulo.

Por último, apresentou-se o monitoramento de alguns recursos, como, por exemplo, de memória e da utilização de CPU para demostrar a utilização da ArqDep. Ao utilizar a ArqDep, o administrador da infraestrutura foi capaz de monitorar os recursos computacionais, auxiliando assim a tomada de decisões.

Capítulo 5

Considerações Finais

Esta Dissertação de Mestrado apresentou uma arquitetura de Computação em Nuvem, denominada ArqDep. A ArqDep apresenta aspectos de dependabilidade para os sistemas de Computação em Nuvem. No decorrer do trabalho, foram apresentados conceitos de Computação em Nuvem e da dependabilidade. A Computação em Nuvem tornou-se um modelo de negócio viável para as grandes, médias e pequenas empresas de TI, pois os provedores de serviços nas nuvens oferecem recursos computacionais escaláveis e o pagamento é realizado através da bilhetagem, ou seja, somente pelo uso. As empresas deixaram de investir grandes quantias em equipamentos de infraestrutura, tornando o investimento menor para iniciar os seus negócios.

A ArqDep demonstrou apta para ser utilizada em relação ao seu objetivo, que era realizar o monitoramento contínuo da infraestrutura de Computação em Nuvem. Nessa Dissertação de Mestrado foi mostrado o monitoramento da utilização de CPU e do uso de memória. Utilizando a ArqDep os usuários e os administradores da infraestrutura são capazes de monitorar os recursos computacionais durante toda a vida útil do sistema. Utilizando o componente proposto para o monitoramento da infraestrutura, o *Ceilometer*, é possível inserir *plugins* que realizem o monitoramento de diversos componentes de acordo com a necessidade dos usuários. Estes *plugins* são escritos em *Python* e são incorporados à infraestrutura.

Uma infraestrutura de Computação em Nuvem foi projetada, para implementar a arquitetura. O modelo de serviço usado nesta Dissertação de Mestrado foi o IaaS, em uma nuvem privada. Para auxiliar na implementação da infraestrutura foi utilizado o *software* de código aberto de gerenciamento de infraestrutura de Computação em Nuvem o *OpenStack*.

Para realizar dos experimentos, foram reproduzidas situações do dia a dia dos usuários de serviços de Computação em Nuvem. Foram criadas diversas instâncias de máquinas virtuais na infraestrutura, a partir de diferentes imagens. Essas instâncias foram monitoradas e foram coletadas métricas de diversos recursos que afetam o desempenho, a eficiência, a disponibilidade e confiabilidade dos sistemas computacionais.

Uma contribuição desta Dissertação de Mestrado foi a construção do módulo de gerenciamento de infraestrutura de Computação em Nuvem. O módulo de gerenciamento é responsável pelo monitoramento e pelo controle da infraestrutura. O módulo de gerenciamento é capaz de criar relatórios gerenciais de forma automatizada aos usuários do provedor de Computação em Nuvem, por meio do monitoramento dos componentes da infraestrutura. O controle tem a função de realizar a classificação das falhas e a realocação de recursos. Por meio de sensores acoplados no sistema de monitoramento, dados são coletados e enviados a camada de controle.

Diversos sensores podem ser instalados no módulo, para demostração foi instalado o sensor de análise de falhas, que captura as informações dos *logs* e enviam para a camada de controle. A camada de controle é responsável pela classificação das falhas, que se baseia no modelo de classificação proposta também nesta Dissertação de Mestrado e pela realocação de recursos. Utilizando este modelo de classificação de falhas é construída uma Árvore de falhas do sistema. O modelo de classificação de falhas também é uma contribuição desta Dissertação de Mestrado. Na literatura não tem uma classificação que contemple de forma detalhada as possíveis falhas em ambientes de Computação em Nuvem, como no modelo de classificação proposto.

A realocação de recursos é função da camada de controle do módulo de gerenciamento. Utilizando o algoritmo proposto de realocação de recursos o sistema realiza a realocação de recursos entre as máquinas virtuais da infraestrutura aumentando o desempenho do sistema, a confiabilidade e a disponibilidade dos serviços. A realocação de recursos é baseada no SLA assinado entre o provedor de serviços e seus consumidores. Este algoritmo é uma contribuição importante, pois aumenta a eficiência e diminui o custo do sistema.

No decorrer de todo o trabalho, foi possível concluir que a Computação em Nuvem pode-se beneficiar de ferramentas e conceitos de dependabilidade para prover serviços cada vez mais confiável aos seus consumidores. Um ambiente de Computação em Nuvem, mesmo em nuvens privadas não é fácil de gerenciar, devido à sua heterogeneidade dos recursos e dos componentes computacionais. Logo, o monitoramento dos recursos computacionais ganha um destaque importante, pois por meio desse monitoramento diversas decisões relativas ao SLA podem ser tomadas.

Diversas atividades podem serem feitas para melhorar este trabalho. Como sugestões de melhoria pode-se citar a construção de um modelo de análise da dependabilidade. Este modelo deve contemplar análises estatísticas do sistema de uma forma geral. E a implementação do algoritmo de realocação de recursos parametrizado.

A infraestrutura construída pode ser utilizada pela instituição para novas pesquisas voltadas ao tema de Computação em Nuvem. Pesquisas nos modelos de serviços SaaS e PaaS podem utilizar a infraestrutura desenvolvida para desenvolver aplicações, implantar aplicações, realizar testes e diversas outras pesquisas.

Como trabalhos futuros a esta pesquisa destacam-se o desenvolvimento de um me-

canismo de coleta automatizada de falhas, sem precisar analisar os *logs* manualmente. Também pode ser feito como trabalho futuro a criação e a implementação de algoritmos para a realocação de recursos que contemple todas as formas de limitar os recursos no SLA, visto que somente o algoritmo de limite máximo fixo foi desenvolvido. Pesquisas para prover uma maior segurança a infraestrutura deve ser realizada. Protocolos de segurança devem ser desenvolvidos, possibilitando o aumento na segurança das informações contidas na infraestrutura.

Referências

[Amazon 2013] Amazon (2013). Amazon Elastic Compute Cloud (Amazon EC2). URL: `http://aws.amazon.com/pt/ec2/`. Acesso em 14/10/2013.

[Armbrust et al. 2009] Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R. H., Konwinski, A., Lee, G., Patterson, D. A., Rabkin, A., Stoica, I., e Zaharia, M. (2009). Above the Clouds: A Berkeley View of Cloud Computing. Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley.

[Avizienis et al. 2004] Avizienis, A., Laprie, J.-C., Randell, B., e Landwehr, C. (2004). Basic concepts and taxonomy of dependable and secure computing. *Dependable and Secure Computing, IEEE Transactions on*, 1(1):11–33.

[Barbetta et al. 2004] Barbetta, P., Reis, M., e Bornia, A. (2004). *Estatística: para cursos de engenharia e informática*. Atlas, 3 edition.

[Bauer e Adams 2012] Bauer, E. e Adams, R. (2012). *Reliability and Availability of Cloud Computing*. John Wiley & Sons.

[Bojanova e Samba 2011] Bojanova, I. e Samba, A. (2011). Analysis of Cloud Computing Delivery Architecture Models. In *Advanced Information Networking and Applications (WAINA), 2011 IEEE Workshops of International Conference on*, pp. 453–458.

[Buyya et al. 2009] Buyya, R., Yeo, C. S., Venugopal, S., Broberg, J., e Brandic, I. (2009). Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Gener. Comput. Syst.*, 25(6):599–616.

[Cannon et al. 2011] Cannon, D. L., Taylor, S., e Britain, G. (2011). *ITIL service operation*. TSO.

[Cearley e Phifer 2009] Cearley, D. e Phifer, G. (2009). Case studies in cloud computing. *Gartner, URL: http://www. gartner. com/it/content/1286700/1286717/march_4_case_studies_in_cloud_computing_dcearley_gphifer. pdf*.

[Ciurana 2009] Ciurana, E. (2009). *Developing with Google App Engine*. Apressus Series. Apress, 1 edition.

[Dai et al. 2009] Dai, Y.-S., Yang, B., Dongarra, J., e Zhang, G. (2009). Cloud service reliability: Modeling and analysis. In *The 15th IEEE Pacific Rim International Symposium on Dependable Computing*.

[de Araújo Macêdo et al. 2010] de Araújo Macêdo, R. J., Barreto, M. E., de Sá, A. S., Freitas, A. E. S., e Almeida, L. (2010). Suporte à Confiança no Funcionamento em Ambientes Federados de Computação nas Nuvens. *Projeto JiT Clouds Anexo A Relatórios de Levantamento de Estado da Arte*, p. 72.

[de São Paulo 2011] de São Paulo, O. E. (2011). Problemas na Amazon tiram empresas de internet do ar. URL: `http://www.estadao.com.br/noticias/impresso,problemas-na-amazon-tiram-empresas-de-internet-do-ar,709493,0.htm`. Acesso em 26/08/2013.

[DeveloperWorks 2013] DeveloperWorks, I. (2013). Emulação do Sistema com o QEMU. URL: `http://www.ibm.com/developerworks/br/library/l-qemu/`. Acesso em 30/08/2013.

[Dillon et al. 2010] Dillon, T., Wu, C., e Chang, E. (2010). Cloud computing: Issues and challenges. In *Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on*, pp. 27–33. Ieee.

[Educação 2011] Educação, U. (2011). Haddad culpa área técnica por falhas no Sisu. URL: `http://educacao.uol.com.br/noticias`. Acesso em 26/08/2013.

[Elmroth e Larsson 2009] Elmroth, E. e Larsson, L. (2009). Interfaces for Placement, Migration, and Monitoring of Virtual Machines in Federated Clouds. In *Grid and Cooperative Computing, 2009. GCC '09. Eighth International Conference on*, pp. 253–260.

[Eugster et al. 2003] Eugster, P. T., Felber, P. A., Guerraoui, R., e Kermarrec, A.-M. (2003). The Many Faces of Publish/Subscribe. *ACM Comput. Surv.*, 35(2):114–131.

[Foster et al. 2008] Foster, I., Zhao, Y., Raicu, I., e Lu, S. (2008). Cloud Computing and Grid Computing 360-Degree Compared. In *Proceedings of the Grid Computing Environments Workshop, 2008. GCE '08*, pp. 1–10.

[Fouquet et al. 2009] Fouquet, M., Niedermayer, H., e Carle, G. (2009). Cloud computing for the masses. In *Proceedings of the 1st ACM workshop on User-provided networking: challenges and opportunities*, U-NET '09, pp. 31–36, Rome, Italy, New York, NY, USA. ACM.

[Francês 2003] Francês, C. R. L. (2003). Introdução às Redes de Petri. *Laboratório de Computação Aplicada, Universidade Federal do Pará.*

[Gartner 2009] Gartner (2009). Gartner Says Cloud Consumers Need Brokerages to Unlock the Potential of Cloud Services. URL: `http://www.gartner.com/newsroom/id/1064712`. Acesso em 07/08/2013.

[Google 2014] Google (2014). Google Trends. URL: `http://www.google.com/trends/explore#q=cloud%20computing&cmpt=date`. Acesso em 05/01/2014.

[Haas e Haas] Haas, P. e Haas, P. *Stochastic Petri Nets: Modelling, Stability, Simulation.* Operations research.

[Haverkort 2002] Haverkort, B. R. (2002). Lectures on Formal Methods and Performance Analysis. chapter Markovian Models for Performance and Dependability Evaluation, pp. 38–83. Springer-Verlag New York, Inc., New York, NY, USA.

[Huang et al. 2013] Huang, J., Lin, C., Kong, X., Wei, B., e Shen, X. (2013). Modeling and Analysis of Dependability Attributes of Services Computing Systems. *Services Computing, IEEE Transactions on*, (99):1–1.

[InfoEscola 2013] InfoEscola (2013). Espionagem dos EUA pela internet. URL: `http://www.infoescola.com/atualidades/espionagem-dos-eua-pela-internet/`. Acesso em 20/10/2013.

[ISO 2013] ISO (2013). ISO 9000 - Quality management. URL: `http://www.iso.org/iso/home/standards/management-standards/iso_9000.htm`. Acesso em 15/08/2013.

[Jansen e Grance 2011] Jansen, W. e Grance, T. (2011). Guidelines on security and privacy in public cloud computing. *NIST special publication*, pp. 800–144.

[Johnson 1989] Johnson, B. (1989). *Design and Analysis of Fault-tolerant Digital Systems*. Addison-Wesley series in electrical and computer engineering. Addison-Wesley Publishing Company.

[Júnior et al. 1996] Júnior, J., do Valle Pereira, V., e de Santa Catarina. Centro Tecnológico, U. F. (1996). *Confiabilidade e falhas de campo: um estudo de caso para melhoria da confiabilidade de um produto e do reparo, através de um procedimento sistemático de coleta de dados*.

[Kuo e Zuo 2003] Kuo, W. e Zuo, M. (2003). *Optimal Reliability Modeling: Principles and Applications*. Wiley.

[Liu et al. 2011] Liu, F., Tong, J., Mao, J., Bohn, R., Messina, J., Badger, L., e Leaf, D. (2011). NIST Cloud Computing Reference Architecture. *NIST special publication*, 500:292.

[Marilly et al. 2002] Marilly, E., Martinot, O., Papini, H., e Goderis, D. (2002). Service level agreements: a main challenge for next generation networks. In *Universal Multiservice Networks, 2002. ECUMN 2002. In Proceedings 2nd European Conference on*, pp. 297–304.

[Matias et al. 2013] Matias, Jr., R., Oliveira, G. D., e de Araujo, L. B. (2013). In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, pp. 1644–1649.

[Mell e Grance 2011] Mell, P. e Grance, T. (2011). The NIST definition of cloud computing (draft). *NIST special publication*, 800(145):7.

[Menasce] Menasce, D.A. e Almeida, V. *Planejamento de Capacidade para Serviços na Web: Métrica, Modelos e Métodos*. CAMPUS - BB, 1 edition.

[Muppala et al. 2000] Muppala, J. K., Fricks, R. M., e Trivedi, K. S. (2000). Techniques for System Dependability Evaluation. *INTERNATIONAL SERIES IN OPERATIONS RESEARCH AND MANAGEMENT SCIENCE*, 154(24):445–479.

[Musa et al. 1987] Musa, J. D., Iannino, A., e Okumoto, K. (1987). *Software reliability: measurement, prediction, application.* McGraw-Hill series in software engineering and technology. McGraw-Hill, Inc., New York, NY, USA.

[Nurmi et al. 2009] Nurmi, D., Wolski, R., Grzegorczyk, C., Obertelli, G., Soman, S., Youseff, L., e Zagorodnov, D. (2009). The Eucalyptus Open-Source Cloud-Computing System. In *In Cluster Computing and the Grid, 2009. CCGRID '09. 9th IEEE/ACM International Symposium on*, pp. 124–131.

[OpenNebula 2013] OpenNebula (2013). An Overview of OpenNebula. URL: `http://docs.opennebula.org/stable/design_and_installation/getting_started/intro.html`. Acesso em 12/08/2013.

[OpenStack 2012] OpenStack (2012). Welcome to the Ceilometer developer documentation! URL: `http://ceilometer.readthedocs.org/en/latest/index.html`. Acesso em 12/09/2013.

[OpenStack 2013a] OpenStack (2013a). Open source software for building private and public clouds. URL: `http://www.openstack.org/`. Acesso em 26/09/2013.

[OpenStack 2013b] OpenStack (2013b). OpenStack Compute Administration Guide. URL: `http://docs.openstack.org/grizzly/openstack-compute/admin/bk-compute-adminguide-grizzly.pdf`. Acesso em 12/09/2013.

[OpenStack 2013c] OpenStack (2013c). Openstack installation Guide for Ubuntu 12.04 (LTS)- HAVANA. URL: `http://docs.openstack.org/trunk/install-guide/install/apt/content/ch_preface.html`. Acesso em 27/10/2013.

[OpenStack 2013d] OpenStack (2013d). Welcome to the Ceilometer developer documentation. URL: `docs.openstack.org/developer/ceilometer/`. Acesso em 25/09/2013.

[Opscode 2013] Opscode (2013). Chef. URL: `http://www.opscode.com/chef/`. Acesso em 01/12/2013.

[Parhami 1988] Parhami, B. (1988). From defects to failures: a view of dependable computing. *ACM SIGARCH Computer Architecture News*, 16(4):157–168.

[QEMU 2013] QEMU (2013). QEMU Open Source Processor Emulator. URL: `http://wiki.qemu.org/Main_Page`. Acesso em 30/08/2013.

[Rackspace 2013] Rackspace (2013). Rackspace the open cloud company. URL: `http://www.rackspace.com/pt/`. Acesso em 07/08/2013.

[Rausand e Høyland 2004] Rausand, M. e Høyland, A. (2004). *System Reliability Theory: Models, Statistical Methods, and Applications.* Wiley Series in Probability and Statistics - Applied Probability and Statistics Section.

[Robinson 2008] Robinson, D. (2008). *Amazon Web Services Made Simple: Learn How Amazon Ec2, S3, Simpledb and Sqs Web Services Enables You to Reach Business Goals Faster.* Emereo Pty Limited, 1 edition.

[Salesforce 2013] Salesforce (2013). CRM e Cloud Computing para crescer seu negócio - Salesforce.com Brasil. URL: `http://salesforce.com`. Acesso em 19/08/2013.

[Schnjakin et al. 2010] Schnjakin, M., Alnemr, R., e Meinel, C. (2010). Contract-based cloud architecture. In *In Proceedings of the second international workshop on Cloud data management*, CloudDB '10, pp. 33–40, Toronto, ON, Canada, New York, NY, USA. ACM.

[Sousa et al. 2009] Sousa, F. R., Moreira, L. O., e Machado, J. C. (2009). Computação em nuvem: Conceitos, tecnologias, aplicações e desafios. *Universidade Federal do Ceará*.

[S.Paulo 2011] S.Paulo, F. D. (2011). Site da PF que agenda passaporte está fora do ar. URL: `http://www1.folha.uol.com.br/cotidiano/977722-site-da-pf-que-agenda-passaporte-esta-fora-do-ar.shtml`. Acesso em 26/08/2013.

[Taurion 2009] Taurion, C. (2009). *Cloud Computing - Computação em Nuvem*. Brasport.

[TecMundo 2011] TecMundo (2011). Serviços da PSN voltam nos próximos dias, atualização do PS3 disponível. URL: `http://www.tecmundo.com.br/sony/10079-servicos-da-psn-voltam-nos-proximos-dias-atualizacao-do-ps3.htm`. Acesso em 26/08/2013.

[Terra 2013] Terra (2013). EUA admitem que espionagem causa "momento de tensão"com aliados. URL: `http://noticias.terra.com.br/mundo/estados-unidos/eua-admitem-que-espionagem-causa-momento-de-tensao-com-aliados,fc22406129be1410VgnCLD2000000ec6eb0aRCRD.html`. Acesso em 01/11/2013.

[Trivedi et al. 1996] Trivedi, K. S., Hunter, S., Garg, S., e Fricks, R. (1996). Reliability Analysis Techniques Explored Through a Communication Network Example.

[Vaquero et al. 2008] Vaquero, L. M., Rodero-Merino, L., Caceres, J., e Lindner, M. (2008). A break in the clouds: towards a cloud definition. *SIGCOMM Comput. Commun. Rev.*, 39(1):50–55.

[Vecchiola et al. 2009] Vecchiola, C., Chu, X., e Buyya, R. (2009). Aneka: A Software Platform for .NET-based Cloud Computing. *CoRR*, abs/0907.4622.

[Veríssimo e de Lemos 1989] Veríssimo, P. e de Lemos, R. (1989). Confiança no funcionamento: Proposta para uma terminologia em português. *Publicação conjunta INESC e LCMI/UFSC*.

[Xie et al. 2004] Xie, M., Dai, Y., e Poh, K. (2004). *Computing System Reliability: Models and Analysis*. Cell Engineering Series. Springer.

Apêndice A

Instalação e configuração do MySQL

Execute o pacote de instalação do MySQL:

```
# apt-get install python-mysqldb mysql-server
```

Edite o arquivo /etc/mysql/my.cnf em *bind-address* para o endereço interno do controlador. Isso permitirá o acesso externo ao nó controlador.

```
# Instead of skip-networking the default is now to listen only on
# localhost which is more compatible and is not less secure.
# bind-address = 200.19.151.16
```

Reinicie o MySQL para efetuar algumas modificações necessárias:

```
# service mysql restart
```

Nos demais nós instale o cliente MySQL e a biblioteca Python MySQL:

```
# apt-get install python-mysqldb
```

Deve-se excluir os usuários anônimos que são criados quando o MySQL inicia pela primeira vez. Esse procedimento deve ser tomado para evitar erros no futuro. Utilize o seguinte comando para realizar esta tarefa:

```
# mysql_secure_installation
```

Este comando apresenta uma série de opções para a instalação com sucesso do banco de dados.

Apêndice B

Instalação e configuração do RabbitMQ

Comando para instalar o servidor *RabbitMQ*:

```
# apt-get install rabbitmq-server
```

Após a instalação dos pacotes do *rabbitmq-server* ele reinicia automaticamente e cria uma senha padrão que deve ser altera para aumentar a segurança do sistema. A senha é alterada utilizando o seguinte comando:

```
# rabbitmqctl change_password guest RABBIT_PASS
```

Apêndice C

Instalação e configuração do *Keystone*

A seguir as instruções para a instalação do *Keystone*. Defini-se também como é realizada a criação de usuários, *tenants*, *tokens* e funções. E por último têm-se a definição dos serviços e *API Endpoints*.

Instale o *Keystone* juntamente com o *phyton-keystoneclient*:

apt-get install keystone

O *Keystone* utiliza o banco de dados para armazenar as suas informações, então é necessário especificar o local do banco de dados no arquivo de configuração. O banco de dados utilizado na infraestrutura é o MySQL, que já foi instalado no controlador. O nome do usuário utilizado é controlle01. Edite o arquivo */etc/keystone/keystone.conf* e altere a seção *[sql]*:

... [sql]
# The SQLAlchemy connection string used to connect to the database
connection = mysql://controller01:KEYSTONE_DBPASS@200.19.151.16/keystone ...

Substitua *KEYSTONE_DBPASS* pela senha do usuário do banco de dados. Deve ser excluído o arquivo *keystone.sqlite* criado em */var/lib/keystone/* para que não seja usado por engano. Isso deve ser feito porque os pacotes do Ubuntu criam por *default* um banco de dados SQLite. Utilizando a senha já definida anteriormente, logue como *root* e crie um usuário no banco de dado*Keystone*.

mysql -u root -p
mysql> CREATE DATABASE keystone;
mysql> GRANT ALL PRIVILEGES ON keystone.* TO 'keystone'@'localhost'
IDENTIFIED BY 'KEYSTONE_DBPASS';

mysql> GRANT ALL PRIVILEGES ON keystone.* TO 'keystone'@'IDENTIFIED BY 'KEYSTONE_DBPASS';

Inicie o serviço *Keystone* e crie as tabelas:

keystone-manage db_sync
service keystone restart

Defina um *token* de autorização para compartilhar com o *Keystone* e outros serviços do OpenStack. Utilize o *openssl* para gerar aleatoriamente o *token* e armazenar no arquivo de configuração:

openssl rand -hex 10

Edite */etc/keystone/keystone.conf*, altere a seção *[DEFAULT]* e substitua *ADMIN_TOKEN* pelo *token* gerado.

[DEFAULT]
# A "shared secret"between keystone and other openstack services
admin_token = ADMIN_TOKEN
...

Reinicie o *Keystone*

service keystone restart

Definindo usuários, *tenants* e funções. Como nenhum usuário ainda foi criado, utiliza-se o *token* criado anteriormente para realizar uma autenticação. Substitua *FCAF3E* pelo *token* gerado. Neste momento é necessário especificar onde o *Keystone* está sendo executado.

export OS_SERVICE_TOKEN=FCAF3E...
export OS_SERVICE_ENDPOINT=http://200.19.151.16:1026

Para criação de *tenant* é necessário no primeiro momento criar um *tenant* para um usuário administrativo e um *tenant* para utilização dos outros serviços do OpenStack.

keystone tenant-create –name=admin –description="Admin Tenant"
keystone tenant-create –name=service –description="Service Tenant"

Em seguida crie um usuário administrativo chamado *admin*. Escolha uma senha para o usuário *admin* e insira um endereço de e-mail para a conta.

keystone user-create –name=admin –pass=ADMIN_PASS
–email=geycy@mestrado.ufu.br

Crie uma função para as tarefas administrativas chamada *admin*. Todas as funções criadas devem ser mapeadas para as funções especificadas nos arquivos *policy.json* dos vários serviços do OpenStack. Por *default* os arquivos *policy* usam a função *admin* para acessar a maioria dos serviços.

keystone role-create –name=admin

Funções são adicionadas aos usuários. Os usuários sempre entram como *tenant* e funções são atribuídas para usuários dentro dos *tenants*. Adicione a função *admin* para o usuário *admin* quando efetuar o *login* com o *tenant admin*.

keystone user-role-add –user=admin –tenant=admin –role=admin

E por último tem a definição dos serviços e dos *API Endpoints*. O *Keystone* controla quais os serviços do OpenStack estão instalados e onde localiza-los na rede. Para cada serviço instalado você chama o *keystone service-create* para descrever o serviço e o *Keystone endpoint-create* para especificar os parâmetros de API associados ao serviço.

keystone service-create –name=keystone –type=identity
–description="Keystone Identity Service"
+————-+———————————+
| Property | Value |
+————-+———————————+
description	Keystone Identity Service
id	15c11a23667e427e91bc31335b45f4bd
name	keystone
type	identity
+————-+———————————+

Especificar um *API Endpoint* para o *Keystone* utilizando o ID de serviço retornado. Quando é especificado um *endpoint*, você fornece URls para API pública, API interna e API administrativa.

```
# keystone endpoint-create
–service-id=the_service_id_above
–publicurl=http://controller:5000
–internalurl=http://controller:5000
–adminurl=http://controller:1026
+————-+———————————+
| Property | Value |
+————-+———————————+
| adminurl | http://200.19.151.16:1026 |
| id | 11f9c625a3b94a3f8e66bf4e5de2679f |
| internalurl | http://200.19.151.16:5000 |
| publicurl | http://200.19.151.16:5000 |
| region | regionOne |
| service_id | 15c11a23667e427e91bc31335b45f4bd |
+————-+———————————+
```

Apêndice D

Instalação e configuração do *Glance*

A seguir a descrição de como foi realizada a instalação e configuração do componente *Glance*. O *Glance* é dividido nos seguintes componentes o *glance-api*, o *glance-registry*, banco de dados e repositório de armazenamento para arquivos de imagens. O comando para instalação do *Glance* é:

apt-get install glance

Em seguida é necessário definir o banco de dados e configurar a sua localização. Edite o arquivo */etc/glance/glance-api.conf* e */etc/glance/glance-registry.conf*, alterações devem ser feitas na seção *[DEFAULT]*, onde aparecer *GLANCE_DBPASS* altere pela sua senha do banco de dados.

...
[DEFAULT]
...
SQLAlchemy connection string for the reference implementation
registry server. Any valid SQLAlchemy connection string is fine.
See: http://www.sqlalchemy.org/docs/05/reference/sqlalchemy/connections.
html#sqlalchemy.create_engine
sql_connection = mysql://
controller01:GLANCE_DBPASS@200.19.151.16/glance
...

Deve ser excluído o arquivo *glance.sqlite* criado em */var/lib/glance/* para que não seja usado por engano. Isso deve ser feito porque os pacotes do Ubuntu criam por DEFAULT um banco de dado SQLite. Utilizando a senha já definida anteriormente, logue como *root* e crie um banco de dados *Glance*.

```
# mysql -u root -p
mysql> CREATE DATABASE glance;
mysql> GRANT ALL PRIVILEGES ON glance.* TO 'glance'@'localhost'
IDENTIFIED BY 'GLANCE_DBPASS';
mysql> GRANT ALL PRIVILEGES ON glance.* TO 'glance'@'%'
IDENTIFIED BY 'GLANCE_DBPASS';
```

Crie as tabelas no banco de dados para o *Glance*:

```
# glance-manage db_sync
```

Após crie um usuário *Glance* que pode ser autenticado com o *Keystone*. Escolha uma senha e um e-mail para este usuário. Utilize o *tenant service* e coloque a função *admin* ao usuário:

```
# keystone user-create –name=glance –pass=GLANCE_PASS
–email=geycy@mestrado.ufu.br
# keystone user-role-add –user=glance –tenant=service –role=admin
```

Em seguida adicione as credencias para o *Glance* configurando os seguintes arquivos: */etc/glance/glance-api.conf* e *etc/glance/glance-registry.conf*. Altere a seção *[keystone_ authtoken]*:

```
...
[keystone_ authtoken]
auth_host = controller01
auth_port = 1026
auth_protocol = http
admin_tenant_name = service
admin_user = glance
admin_password = GLANCE_PASS
...
```

É necessário também adicionar as credencias nos seguintes arquivos */etc/glance/glance-api-paste.ini* e */etc/glance/glance-registry-paste.ini*. Alterando em cada arquivo na seção *[filter:authtoken]*:

```
[filter:authtoken]
paste.filter_factory=keystoneclient.middleware.auth_token:filter_factory
auth_host=controller
```

admin_user=glance
admin_tenant_name=service
admin_password=GLANCE_PASS

É importante registrar o *Glance* com o *Keystone* para que os outros serviços do OpenStack possa localizá-lo. O próximo comando registra o serviço e cria um *endpoint*:

keystone service-create –name=glance –type=image
–description="Glance Image Service"

O ID retornado é usado para criar o *endpoint*:

keystone endpoint-create
–service-id=the_service_id_above
–publicurl=http://200.19.151.16:9292
–internalurl=http://200.19.151.16:9292
–adminurl=http://200.19.151.16:9292

E finalmente reinicie o *Glance*:

service glance-registry restart
service glance-api restart

Apêndice E

Instalação e configuração do *Nova*

A seguir as instruções para a instalação e configuração dos componentes do *Nova*. Os componentes do *Nova* estão divididos entre a Máquina Controlador e a Máquina *Compute*. A maioria dos componentes estão no Controlador, no *Compute* concentra os componentes responsáveis pela inicialização e execução das máquinas virtuais. Vamos iniciar pelos componentes instalados no Controlador.

Primeiro é necessário instalar os pacotes do *Nova*, esses pacotes que executam os serviços do *Nova* no Controlador:

apt-get install nova-novncproxy novnc nova-api
nova-cert
nova-consoleauth nova-scheduler
python-novaclient

As informações no *Nova* são armazenadas em um banco de dados. O banco de dados utilizado na nossa infraestrutura é o *MySQL*. Edite o arquivo */etc/nova/nova.conf* e adicione as linhas nas seções *[database]* e *[keystone _ authtoken]*. Onde aparecer NOVA_ DBPASS substitua pela senha do *Nova*.

...
[database]
# The SQLAlchemy connection string used to connect to the database
connection = mysql://nova:NOVA_ DBPASS@controller/nova
[keystone_ authtoken]
auth_ host = controller
auth_ port = 35357
auth_ protocol = http
admin_ tenant_ name = service

admin_ user = nova
admin_ password = NOVA_ PASS

O *Nova* deve ser configurado para utilizar os serviços de mensagens do *RabbitMQ*. É necessário definir as configurações. O arquivo */etc/nova/nova.conf* deve ser editado:

rpc_ backend = nova.rpc.impl_ kombu
rabbit_ host = controller
rabbit_ password = RABBIT_ PASS

Deve ser excluído o arquivo *nova.sqlite* criado em */var/lib/nova/* para que não seja usado por engano. Isso deve ser feito porque os pacotes do Ubuntu criam por *default* um banco de dados *SQLite*. Utilizando a senha já definida anteriormente, logue como *root* e crie um banco de dados *Nova*:

mysql -u root -p
mysql> CREATE DATABASE nova;
mysql> GRANT ALL PRIVILEGES ON nova.* TO 'nova'@'localhost'
IDENTIFIED BY 'NOVA_ DBPASS';
mysql> GRANT ALL PRIVILEGES ON nova.* TO 'nova'@'%'
IDENTIFIED BY 'NOVA_ DBPASS';

Crie as tabelas no banco de dados para o *Nova*:

nova-manage db sync

Defina as configurações do *my_ ip*, do *vncserver_ listen*, e do *vncserver_ proxyclient_ address* para o endereço de IP do nó Controlador. Edite o arquivo */etc/nova/nova.conf* e adicione as seguintes linhas na seção *[DEFAULT]*:

...
[DEFAULT]
...
my_ ip=ip
vncserver_ listen=ip
vncserver_ proxyclient_ address=ip

Após crie um usuário *Nova* que pode ser autenticado com o *Keystone*. Escolha uma senha e um e-mail para este usuário. Utilize o *tenant service* e der a função *admin* ao

usuário:

keystone user-create –name=nova –pass=NOVA_ PASS–email=nova@example.
com
keystone user-role-add –user=nova –tenant=service –role=admin

Em seguida adicione as credencias para o *Nova* configurando o seguinte arquivo */etc/nova/nova.conf* e adicione na seção *[DEFAULT]* as seguintes linhas:

#...
[DEFAULT]
...
auth_ strategy=keystone

É necessário também adicionar as credenciais no seguinte arquivo */etc/nova/api-paste.ini* e adicione as seguintes opções na seção *[filter:authtoken]*:

[filter:authtoken]
paste.filter_ factory=keystoneclient.middleware.auth_ token:filter_ factory
auth_ host=controller
auth_ port=5000
auth_ protocol=http
auth_ uri=http://controller:5000/v2.0
admin_ tenant_ name=service
admin_ user=nova
admin_ password=NOVA_ PASS

É importante registar o *Nova* com o *Keystone* para os outros serviços do *OpenStack* possa localizá-lo. O próximo comando registra o serviço e cria um *endpoint*:

keystone service-create –name=nova –type=compute
–description="Nova Compute service"

O ID retornado é usado para criar o *endpoint*:

keystone endpoint-create
–service-id=the_ service_ id_ above
–publicurl=http://controller:8774/v2/%*tenant_id*s
–internalurl=http://controller:8774/v2/%*tenant_id*s

–adminurl=http://controller:8774/v2/%*tenant_id*s

E finalmente reinicie o *Nova*:

```
# service nova-api restart
# service nova-cert restart
# service nova-consoleauth restart
# service nova-scheduler restart
# service nova-novncproxy restart
```

Outros componentes do *Nova* foram instalados no *Compute*. O nó *Compute* recebe as solicitações do nó *Controlador* e armazena as instâncias de máquinas virtuais. A opção de separar o *Compute* e o *Controlador* é pela facilidade de escalar horizontalmente a infraestrutura. Do modo que foi configurada, pode-se adicionar nós *Compute* a infraestrutura a qualquer momento. A seguir os comandos para a instalação dos componentes presentes no nó *Compute*.

Primeiro é necessário configurar as interfaces de rede no nó e escolher o *hypervisor* que a infraestrutura utilizará para executar as instâncias de máquinas virtuais. O *hypervisor* utilizado é o QEMU.

```
# nano -w /etc/network/interfaces
#Interface para busca de pacotes na Internet
auto eth0
iface eth0 inet dhcp

# Interface Pública
auto eth0
iface eth0 inet static
address
netmask
network
broadcast

# Interface Privada
auto eth1
iface eth1 inet manual
up ifconfig eth1 up
```

Reinicie as interfaces de rede.

/etc/init.d/networking restart

apt-get install bridge-utils ntp nova-compute nova-network qemu open-iscsi

Edite o arquivo */etc/nova/nova.conf* e adicione as linhas nas seções:

...
[DEFAULT]
...
auth_ strategy=keystone
...
[database]
# The SQLAlchemy connection string used to connect to the database
connection = mysql://nova:NOVA_ DBPASS@controller/nova

O nó *Compute* deve ser configurado para utilizar os serviços de mensagens do *RabbitMQ*. É necessário definir estas configurações. O arquivo */etc/nova/nova.conf* deve ser editado:

rpc_ backend = nova.rpc.impl_ kombu
rabbit_ host = controller
rabbit_ password = RABBIT_ PASS

Defina as configurações do *my_ ip*, do *vncserver_ listen*, e do *vncserver_ proxyclient_ address* para o endereço de IP do nó *Compute*. Edite o arquivo */etc/nova/nova.conf* e adicione as seguintes linhas na seção *[DEFAULT]*:

[DEFAULT]
...
my_ ip= ip
vnc_ enabled=True
vncserver_ listen=0.0.0.0
vncserver_ proxyclient_ address= ip
novncproxy_ base_ url=http://controller:6080/vnc_ auto.html

Especifique o nó que executa o *Glance*. Edite o arquivo */etc/nova/nova.conf* e adicione as linhas abaixo na seção [DEFAULT]:

```
[DEFAULT]
...
glance_ host=controller
```

É necessário também adicionar as credenciais no seguinte arquivo */etc/nova/api-paste.ini* e adicione as seguintes opções na seção *[filter:authtoken]*:

```
[filter:authtoken]
paste.filter_ factory=keystoneclient.middleware.auth_ token:filter_ factory
auth_ host=controller
auth_ port = 35357
auth_ protocol = http
admin_ user=nova
admin_ tenant_ name=service
admin_ password=NOVA_ PASS
```

Reinicie o *Compute*:

```
# service nova-compute restart
```

Instalando os pacotes apropriados do *nova-network*:

```
# apt-get install nova-network
```

Edite o arquivo */etc/nova/nova.conf* para definir as configurações de rede e adicione as seguintes linhas na seção [DEFAULT]:

```
[DEFAULT]

...
network_ manager=nova.network.manager.FlatDHCPManager
firewall_ driver=nova.virt.libvirt.firewall.IptablesFirewallDriver
network_ size=254
allow_ same_ net_ traffic=False
multi_ host=True
send_ arp_ for_ ha=True
share_ dhcp_ address=True
force_ dhcp_ release=True
flat_ network_ bridge=br100
```

```
flat_ interface=eth0
public_ interface=eth1
rabbit_ host=controller

# service nova-network restart
```

Apêndice F

Instalação e configuração do *Swift*

Instale os pacotes do *Swift* e o *openSSH*:

apt-get install swift openssh-server rsync memcached python-netifaces python-xattr python-memcache

Criar e preencha os diretórios de configuração em todos os nós:

mkdir -p /etc/swift
chown -R swift:swift /etc/swift/

Crie o */etc/swift/swift.conf* em todos os nós:

[swift-hash]
random unique string that can never change (DO NOT LOSE)
swift_ hash_ path_ suffix = fLIbertYgibbitZ

apt-get install swift-account swift-container swift-object xfsprogs

Em cada nó que utilizar para armazenamento, o volume deve estar formatado em XFS:

fdisk /dev/sdb
mkfs.xfs /dev/sdb1
echo "/dev/sdb1 /srv/node/sdb1 xfs noatime,nodiratime,nobarrier,logbufs= 8 0 0»> /etc/fstab
mkdir -p /srv/node/sdb1
mount /srv/node/sdb1
chown -R swift:swift /srv/node

Crie o arquivo */etc/rsyncd.conf*:

uid = swift
gid = swift
log file = /var/log/rsyncd.log
pid file = /var/run/rsyncd.pid
address = <STORAGE_ LOCAL_ NET_ IP>
[account]
max connections = 2
path = /srv/node/
read only = false
lock file = /var/lock/account.lock
[container]
max connections = 2
path = /srv/node/
read only = false
lock file = /var/lock/container.lock
[object]
max connections = 2
path = /srv/node/
read only = false
lock file = /var/lock/object.lock

Edite a linha no arquivo */etc/default/rsync*:

RSYNC_ ENABLE = true

Inicie o *daemon rsync*:

service rsync start

Crie um diretório de cache *Swift* e defina sua permissões:

mkdir -p /var/swift/recon
chown -R swift:swift /var/swift/recon

Instale o componente *swift-proxy*:

apt-get install swift-proxy memcached python-keystoneclient python-swiftclient python-

webob

Crie o *self-signed cert* para *SSL*:

```
# cd /etc/swift
# openssl req -new -x509 -nodes -out cert.crt -keyout cert.key
```

Edite o arquivo */etc/memcached.conf* para modificar a memcache:

-l <PROXY_ LOCAL_ NET_ IP>

```
# service memcached restart
```

Comandos utilizados somente para usuários de Ubuntu, pois os pacotes da distribuição não inclui o *Keystoneauth*, deve-se verificar se estão incluídos no *proxy server*:

```
# git clone https://github.com/openstack/swift.git
# cd swift
# python setup.py install
# swift-init proxy start
```

Crie o arquivo */etc/swift/proxy-server.conf*:

```
[DEFAULT]
bind_ port = 8888
user = swift
[pipeline:main]
pipeline = healthcheck cache authtoken keystoneauth proxy-server
[app:proxy-server]
use = egg:swift# proxy
allow_ account_ management = true
account_ autocreate = true
[filter:keystoneauth]
use = egg:swift# keystoneauth
operator_ roles = Member,admin,swiftoperator
[filter:authtoken]
paste.filter_ factory = keystoneclient.middleware.auth_ token:filter_ factory
# Delaying the auth decision is required to support token-less
# usage for anonymous referrers ('.r:*')
```

delay_ auth_ decision = true
cache directory for signing certificate
signing_ dir = /home/swift/keystone-signing
auth_ * settings refer to the Keystone server
auth_ protocol = http
auth_ host = 192.168.56.3
auth_ port = 35357
the same admin_ token as provided in keystone.conf
admin_ token = 012345SECRET99TOKEN012345
the service tenant and swift userid and password created in Keystone
admin_ tenant_ name = service
admin_ user = swift
admin_ password = swift
[filter:cache]
use = egg:swift# memcache
[filter:catch_ errors]
use = egg:swift# catch_ errors
[filter:healthcheck]
use = egg:swift# healthcheck

Crie o *signing_ dir* e o conjunto de permissões:

mkdir -p /home/swift/keystone-signing
chown -R swift:swift /home/swift/keystone-signing

Crie *rings* de conta, *container* e objetos:

cd /etc/swift
swift-ring-builder account.builder create
swift-ring-builder container.builder create
swift-ring-builder object.builder create

Para cada dispositivo de armazenamento *Swift*, adicionar entradas para cada *ring*:

swift-ring-builder account.builder add z<ZONE>-<STORAGE_ LOCAL_ NET_ IP>:6002[R<STORAGE_ REPLICATION_ NET_ IP>:6005]/<DEVICE>100
swift-ring-builder container.builder add z<ZONE>-<STORAGE_ LOCAL_ NET_ IP_ 1>:6001[R<STORAGE_ REPLICATION_ NET_ IP>:6004]/<DEVICE>100
swift-ring-builder object.builder add z<ZONE>-<STORAGE_ LOCAL_ NET_ IP_

1>:6000[R<STORAGE_ REPLICATION_ NET_ IP>:6003]/<DEVICE>100

Verifique o conteúdo de cada *ring*:

```
# swift-ring-builder account.builder
# swift-ring-builder container.builder
# swift-ring-builder object.builder
```

Balancei os *rings*:

```
# swift-ring-builder account.builder rebalance
# swift-ring-builder container.builder rebalance
# swift-ring-builder object.builder rebalance
```

Copie os arquivos *account.ring.gz*, *container.ring.gz*, e *object.ring.gz* para cada nó *Swift* em */etc/swift*.

Certifique que o *Swift* tenha todos os arquivos de configuração:

```
# chown -R swift:swift /etc/swift
```

Inicie o *Proxy-service*:

```
# service proxy-server start
```

Apêndice G

Instalação e configuração do *Cinder*

Os comandos a seguir são usados para instalar e configurar o componente *Cinder* do *OpenStack*. A primeira ação a ser realiza é instalar os pacotes do *Cinder*:

apt-get install cinder-api cinder-scheduler

Edite o arquivo */etc/cinder/cinder.conf* e adicione as linhas nas seções *[database]*:

[database]
...
# The SQLAlchemy connection string used to connect to the
# database (string value)
connection = mysql://cinder:CINDER_ DBPASS@localhost/cinder
...

o banco de dados deve estar configurado para os serviços do *Cinder*. Troque *CINDER_ DBPASS* pela sua senha. Utilizando a senha já definida anteriormente, logue como *root* e crie um banco de dados *Cinder*:

mysql -u root -p
mysql> CREATE DATABASE cinder;
mysql> GRANT ALL PRIVILEGES ON cinder.* TO 'cinder' 'localhost'
IDENTIFIED BY 'CINDER_ DBPASS';
mysql> GRANT ALL PRIVILEGES ON cinder.* TO 'cinder' '%'
IDENTIFIED BY 'CINDER_ DBPASS';

Crie as tabelas no banco de dados para o *Cinder*:

cinder-manage db sync

Após crie um usuário *Cinder* que pode ser autenticado com o *Keystone*. Escolha uma senha e um e-mail para este usuário. Utilize o *tenant service* e coloque a função *admin* ao usuário:

keystone user-create –name=cinder –pass=CINDER_ PASS–email=cinder@example.com
keystone user-role-add –user=cinder –tenant=service –role=admin

Em seguida adicione as credencias para o *Cinder* configurando o seguinte arquivo */etc/cinder/api-paste.ini*. Altere a seção *[filter:authtoken]*:
[filter:authtoken]
paste.filter_ factory=keystoneclient.middleware.auth_ token:filter_ factory
auth_ host=controller
auth_ port = 35357
auth_ protocol = http
admin_ tenant_ name=service
admin_ user=cinder
admin_ password=CINDER_ PASS

Configure o *Cinder* para utilizar o *RabbitMQ* na troca de mensagens. Edite o arquivo */etc/cinder/cinder.conf* e na seção [DEFAULT] altere por:

rpc_ backend = cinder.openstack.common.rpc.impl_ kombu
rabbit_ host = controller
rabbit_ port = 5672
rabbit_ userid = guest
rabbit_ password = RABBIT_ PASS

É importante registrar o *Cinder* com o *Keystone* para que os outros serviços do *OpenStack* possa localizá-lo. O próximo comando registra o serviço e cria um *endpoint*:

keystone service-create –name=cinder –type=volume
–description="Cinder Volume Service"

O ID retornado é usado para criar o *endpoint*:

keystone endpoint-create
–service-id=the_ service_ id_ above
–publicurl=http://controller:8776/v1/%*tenant_ids*

–internalurl=http://controller:8776/v1/%*tenant_id*s
–adminurl=http://controller:8776/v1/%*tenant_id*s

Reinicie o *Swift*:

service cinder-scheduler restart
service cinder-api restart

Apêndice H

Instalação e configuração do *Ceilometer*

Os componentes principais do *Ceilometer* são instalados no Controlador e os agentes são instalados nos nós *Compute* para realizar o monitoramento dos recursos das máquinas virtuais. Instale os pacotes do *Ceilometer*:

apt-get install ceilometer-api ceilometer-collector ceilometer-agent-central python-ceilometerclient

Em seguida é necessário definir o banco de dados e configurar a sua localização. Edite o arquivo */etc/ceilometer/ceilometer.conf* na seção *[DEFAULT]* faça as alterações necessárias:

...
[DEFAULT]
...
SQLAlchemy connection string for the reference implementation
registry server. Any valid SQLAlchemy connection string is fine.
See: http://www.sqlalchemy.org/docs/05/reference/sqlalchemy/connections.
html#sqlalchemy.create_engine
sql_connection = mysql://
controller01:CEILOMETER_ DBPASS@200.19.151.16/ceilometer
...

Uma chave secreta deve ser definida para compartilhar entre os nós que são monitorados. Use o *openssl* para gerar o *token* aleatório e armazene no arquivo de configuração:

openssl rand -hex 10

Edite o arquivo /etc/ceilometer/ceilometer.conf e altere as configurações da seção [publisher_ rpc]. Altere [ADMIN_ TOKEN] o com o resultado do comando *openssl*.

```
...
[publisher_ rpc]
...
# Secret value for signing metering messages (string value)
metering_ secret = ADMIN_ TOKEN
...
```

Modifique o arquivo */etc/ceilometer/ceilometer.conf* para habilitar a troca de mensagens utilizando o RabbitMQ:

```
rabbit_ host = controller
rabbit_ password = RABBIT_ PASS
```

Após crie um usuário *Ceilometer* que pode ser autenticado com o *Keystone*. Escolha uma senha e um e-mail para este usuário. Utilize o *tenant service* e coloque a função *admin* ao usuário:

```
# keystone user-create –name=ceilometer –pass=CEILOMETER_ PASS–email=ceilometer@example.com
# keystone user-role-add –user=ceilometer –tenant=service –role=admin
```

Em seguida adicione as credencias para o *Ceilometer* configurando o seguinte arquivo */etc/ceilometer/ceilometer.conf*. Altere a seção *[keystone_ authtoken]*:

```
[keystone_ authtoken]
auth_ host = controller
auth_ port = 35357
auth_ protocol = http
admin_ tenant_ name = service
admin_ user = ceilometer
admin_ password = CEILOMETER_ PASS
```

É importante registrar o *Ceilometer* com o *Keystone* para que os outros serviços do *OpenStack* possa localizá-lo. O próximo comando registra o serviço e cria um *endpoint*:

```
# keystone service-create –name=ceilometer –type=metering
–description="Ceilometer Metering Service"
```

O ID retornado é usado para criar o *endpoint*:

```
# keystone endpoint-create
–service-id=the_ service_ id_ above
–publicurl=http://controller:8777/
–internalurl=http://controller:8777/
–adminurl=http://controller:8777/
```

Reinicie o *Ceilometer*:

```
# service ceilometer-agent-central restart
# service ceilometer-api restart
# service ceilometer-collector restart
```

Os agentes que coletam os dados são executados no *Compute*. A seguir detalhes da instalação destes agentes.

```
# apt-get install ceilometer-agent-compute
```

Edite o arquivo */etc/nova/nova.conf* e modifique a seção *[DEFAULT]*:

```
...
[DEFAULT]
...
instance_ usage_ audit=True
instance_ usage_ audit_ period=hour
notify_ on_ state_ change=vm_ and_ task_ state
notification_ driver=nova.openstack.common.notifier.rpc_ notifier
notification_ driver=ceilometer.compute.nova_ notifier
```

As chaves secretas devem ser compartilhadas. Edite o arquivo /etc/ceilometer/ceilometer.conf e configure a seção *[DEFAULT]* com as devidas modificações. Troque *ADMIN_ TOKEN* pelo *token* criado anteriormente.

```
...
[publisher_ rpc]
# Secret value for signing metering messages (string value)
metering_ secret = ADMIN_ TOKEN
```

...

Reinicie o serviço:

service ceilometer-agent-compute restart