UNIVERSIDADE DA BEIRA INTERIOR
Engenharia

Tolerância a Falhas em Infraestrutura de Cloud - Construção de Aplicações com Elevada Disponibilidade

José Danilson dos Reis Ferreira

Dissertação para obtenção do Grau de Mestre em
Engenharia Informática
(2º ciclo de estudos)

Orientador: Prof. Doutora Maria Paula Prata de Sousa

Covilhã, Outubro de 2019

Dissertação elaborada no Instituto de Telecomunicações - Delegação da Covilhã e no Departamento de Informática da Universidade da Beira Interior por José Danilson dos Reis Ferreira, Licenciado em Engenharia de Sistemas e Informática pela Universidade Jean Piaget de Cabo Verde, sob orientação da Doutora Maria Paula Prata de Sousa, Investigadora do Instituto de Telecomunicações e Professora Auxiliar do Departamento de Informática da Universidade da Beira Interior, e submetida à Universidade da Beira Interior para discussão em provas públicas.

Trabalho integrado em projeto parcialmente financiado por:

• Programa Integrado de ICDT "C4 – Centro de Competências em Cloud Computing" com contrato CENTRO-01-0145-FEDER-000019;

• FCT - Fundação para a Ciência e a Tecnologia através do projeto número UID/EEA/50008/2019.

Agradecimentos

Aos meus pais, meus irmãos, minhas irmãs, meus sobrinhos pelo apoio, pela compreensão, pelo incentivo dado. À minha orientadora da dissertação, Professora Doutora Maria Paula Prata de Sousa, por toda a disponibilidade, dedicação, orientação e ajuda que prestou para a materialização deste trabalho. Ao Instituto de Telecomunicações - Delegação da Covilhã pelo apoio. Aos professores do Departamento de Informática da Faculdade de Engenharia. À direção da Escola Secundária de São Domingos. Aos meus colegas da turma e do laboratório, pelo apoio e partilha dos conhecimentos e à Deus por me ter dado vida e saúde.

Dedicatórias

Dedico este trabalho aos meus pais Orlando Ferreira e Elísia dos Reis, aos meus irmãos Paulino Ferreira, Adilson Ferreira, Ivandro Ferreira, Nícia Ferreira, Adalgisa ferreira, Gisele Ferreira e Graça Ferreira, aos meus sobrinhos, Edline Ferreira, Liliane Ferreira, Diego Ferreira e Marlon Ferreira. Em fim à todos os meus familiares, colegas, amigos, professores e a todos aqueles que me apoiaram de uma forma direta ou indireta nesta caminhada.

Resumo

Na atualidade, a computação na cloud é cada vez mais usada, como forma de aceder a recursos de computação à medida das necessidades do utilizador, sem necessidade de grandes investimentos. Como qualquer infraestrutura informática a cloud também está sujeitas a falhas. É nesta perspetiva que se desenvolveu um sistema que tolera as falhas numa infraestrutura da cloud Openstack. Foi implementado um cenário que garante a alta disponibilidade de aplicações web alojadas neste sistema da cloud. Este trabalho na primeira fase apresenta-se uma revisão bibliográfica dos conceitos de computação na cloud, de tolerância a falhas e de alta disponibilidade. Na segunda fase debruça–se na instalação e configuração da infraestrutura de cloud OpenSatck. Após isso, foi desenvolvida uma aplicação web com redundância neste sistema cloud e foi avaliado o seu desempenho sem tolerâncias a falhas e com tolerância a falhas. Concluiu-se que é mais viável ter sistemas com tolerâncias a falhas num sistema de cloud, apesar dos custos que acarretam, a não tê-los.

Palavras-chave

Computação na Cloud, OpenStack, Tolerância a Falhas, Alta Disponibilidade, Replicação.

Abstract

Today, cloud computing is increasingly used as a means of accessing computing resources tailored to user needs without the need for major investment. Like any computing infrastructure the cloud is also subject to failure. It is from this perspective that a system has been developed that tolerates failures in an Openstack cloud infrastructure. A scenario has been implemented that ensures the high availability of web applications hosted on this cloud system. This work in the first phase presents a literature review of the concepts of cloud computing, fault tolerance and high availability. In the second phase it focuses on the installation and configuration of the OpenSatck cloud infrastructure. After that, a redundant web application was developed on this cloud system and its performance without fault tolerance and fault tolerance was evaluated. It was concluded that it is more feasible to have fault tolerant systems in a cloud system, despite the high costs, than not to have a fault tolerant system.

Keywords

Cloud Computing, OpenStack, Fault Tolerance, High Availability, Replication.

Conteúdo

1 Introdução **1**

2 Computação na Cloud **3**
2.1 Introdução . . . 3
2.2 Definição e Características . . . 3
2.3 Modelos de Serviços de Computação na Cloud . . . 4
2.3.1 Infraestrutura como um Serviço-IaaS . . . 5
2.3.2 Plataforma como um Serviço-PaaS . . . 5
2.3.3 Software como um Serviço-SaaS . . . 5
2.4 Tipos de Implementação da Cloud . . . 5
2.4.1 Cloud Privada . . . 5
2.4.2 Cloud Pública . . . 6
2.4.3 Cloud Híbrida . . . 6
2.4.4 Cloud Comunitária . . . 6
2.5 A Framework OpenStack . . . 6
2.5.1 Arquitectura dos Componentes do Openstack . . . 7
2.5.2 Serviços do Openstack . . . 8
2.6 Outras Sistemas de Computação na Cloud . . . 11
2.6.1 Amazon Web Services (AWS) . . . 11
2.6.2 Microsoft Windows Azure Plataform . . . 12
2.6.3 Aneka . . . 12
2.7 Virtualização . . . 13
2.8 Trabalhos Relacionados . . . 14
2.9 Conclusão . . . 15

3 Tolerância a Falhas **17**
3.1 Introdução . . . 17
3.2 Conceitos . . . 17
3.3 Tipos de Falhas . . . 19
3.4 Técnicas de Tolerância a Falhas . . . 20
3.5 Gestão de Tolerância a Falhas na Computação na Cloud . . . 22
3.5.1 Gestão Exclusiva de Tolerância a Falhas . . . 23
3.5.2 Gestão Colaborativa de Tolerância a Falhas . . . 24
3.6 Elevada Disponibilidade . . . 24
3.6.1 Elevada Disponibilidade na Infraestrutura de Cloud OpenStack . . . 25
3.6.2 Ferramentas de Elevada Disponibilidade . . . 26
3.7 Conclusão . . . 31

4 Ambiente Experimental e Ferramentas de Avaliação de Desempenho **33**
4.1 Instalação e Configuração da Cloud OpenStack . . . 33
4.1.1 Ambiente de Instalação . . . 33
4.1.2 Arquitetura e Configuração de Rede . . . 34
4.1.3 Instalação e Configuração dos Nós da Infraestrutura da Cloud OpenStack . 35
4.2 Aplicação Exemplo . . . 36

4.3 Aplicação Exemplo num Cenário de Elevada Disponibilidade 37
4.4 Ferramentas de Avaliação de Desempenho 39
4.4.1 Lado do Cliente 39
4.4.2 Lado do Servidor 40

5 Resultados **43**
5.1 Avaliação do Desempenho sem Tolerância a Falhas 43
5.1.1 Avaliação da "Aplicação Exemplo" 43
5.1.2 Avaliação do "Aplicação Exemplo" com HAProxy 44
5.2 Avaliação do Desempenho da Aplicação Web com Elevada Disponibilidade 46
5.3 Análise da Avaliação dos Cenários 58
5.4 Conclusão 60

6 Conclusão e Trabalho Futuro **61**

Bibliografia **63**

Lista de Figuras

2.1 Modelos de computação na cloud [9]. 4
2.2 Arquitectura dos componentes do Openstack [14]. 7

3.1 Percurso para avaria [35]. 18
3.2 Cronograma para a precaução de um sistema de detecção de falhas [39]. 21
3.3 Arquitectura de computação na cloud [7]. 22
3.4 Gestão de tolerância a falhas na computação na cloud. 22
3.5 Configuração ativo/passivo [37]. 26
3.6 Configuração ativo/ativo [37]. 26
3.7 Servidor HAProxy [43]. 28
3.8 Alta disponibilidade de servidores ”HAProxy” com ”Keepalived” : cenário sem falhas (adpatado de [45] [46]). 29
3.9 Alta disponibilidade de servidores ”HAProxy” com ”Keepalived”: cenário após falha do “HAProxy“ “Master” (adpatado de [45] [46]). 29
3.10 Replicação dos servidores do MySQL (adaptado de [47]). 30
3.11 Pacemaker [48]. 31

4.1 Arquitetura e configuração de rede. 34
4.2 Cenário da aplicação na cloud. 37
4.3 Cenário de Alta Disponibilidade. 38
4.4 Interface do Webpagetest [52]. 40
4.5 Exemplo do comando no Apache Bench. 40
4.6 Exemplo do Output do Apache Bench. 41

5.1 Avaliação do desempenho (lado do servidor) sem tolerância a falhas. 44
5.2 Resposta do servidor sem tolerância a falhas. 44
5.3 Cenário sem tolerância a falhas com HAProxy. 45
5.4 Avaliação do desempenho (lado do servidor) sem tolerância a falhas com HAProxy. 45
5.5 Resposta do servidor sem tolerância a falhas com HAProxy. 46
5.6 Avaliação do desempenho (lado do servidor) com alta disponibilidade: cenário sem falhas. 47
5.7 Ambiente de alta disponibilidade sem falhas. 48
5.8 Falha do servidor web da máquina virtual 1. 48
5.9 Avaliação do desempenho (lado do servidor) com alta disponibilidade: cenário com falhas do servidor web da máquina virtual 1. 49
5.10 Ambiente da alta disponibilidade (falha do servidor web da VM 1). 49
5.11 Falha do servidor web da máquina virtual 2. 50
5.12 Avaliação do desempenho (lado do servidor) com alta disponibilidade: cenário com falhas do servidor web da máquina virtual 2. 51
5.13 Ambiente da alta disponibilidade (falha do servidor web da VM 2). 51
5.14 Falha do servidor MySQL “Slave” da máquina virtual 2. 52
5.15 Avaliação do desempenho (lado do servidor) com alta disponibilidade: cenário com falhas do servidor MySQL da máquina virtual 2. 52

5.16 Ambiente da alta disponibilidade (falha do servidor MySQL “Slave” da máquina virtual 2). 53
5.17 Falha do servidor MySQL “Master” da máquina virtual 1. 53
5.18 Avaliação do desempenho (lado do servidor) com alta disponibilidade: cenário com falhas do servidor MySQL da máquina virtual 1. 54
5.19 Ambiente da alta disponibilidade (falha do servidor MySQL “Master” da máquina virtual 1). 54
5.20 Falha do servidor web da VM1 e do servidor MySQL “Slave” da VM2. 55
5.21 Avaliação do desempenho (lado do servidor) com alta disponibilidade: cenário com falha do servidor web da máquina virtual 1 e do servidor MySQL da máquina virtual 2. 56
5.22 Ambiente da alta disponibilidade (falha do servidor web da VM 1 e do servidor MySQl ”Slave” da VM 2). 56
5.23 - Falha do servidor web da máquina virtual 2 e do servidor MySQL “Master” da máquina virtual 1. 57
5.24 Avaliação do desempenho (lado do servidor) com alta disponibilidade: cenário com falha do servidor web da máquina virtual 2 e do servidor MySQL da máquina virtual 1. 57
5.25 Ambiente da alta disponibilidade(falha do servidor web da VM 2 e do servidor MySQl ”Master” da VM 1). 58
5.26 Cenários sem falhas. 59
5.27 Cenários com apenas uma falha. 59
5.28 Cenários com duas falhas. 60

Lista de Tabelas

3.1 Mecanismo para mascarar falhas (Adaptado de [36]). 21

4.1 Características e dados da configuração da máquina 1 (“Controller”). 33
4.2 Características e dados da configuração da máquina 2 (“Compute”). 33
4.3 Caraterísticas e dados da configuração da máquina 3 (“Block Storage”). 34
4.4 Características e dados da configuração máquina virtual 1 (VM 1). 37
4.5 Características e dados da configuração máquina virtual 2 (VM 2). 39

5.1 Avaliação do desempenho (lado do cliente) sem tolerância a falhas. 44
5.2 Avaliação do desempenho (lado do cliente) sem tolerância a falhas com HAProxy. 45
5.3 Avaliação do desempenho (lado do cliente) com alta disponibilidade: cenário sem falhas 47
5.4 Avaliação do desempenho (lado do cliente) com alta disponibilidade: cenário com falha do servidor web da máquina virtual 1 49
5.5 Avaliação do desempenho (lado do cliente) com alta disponibilidade:cenário com falha do servidor web da máquina virtual 2 50
5.6 Avaliação do desempenho (lado do cliente) com alta disponibilidade: cenário com falha do MySQL da máquina virtual 2 51
5.7 Avaliação do desempenho (lado do cliente) com alta disponibilidade: cenário com falha do MySQL da máquina virtual 1. 54
5.8 Avaliação do desempenho (lado do cliente) com alta disponibilidade: cenário com falha do servidor web da máquina virtual 1 e do servidor MySQL da máquina virtual 2. 55
5.9 Avaliação do desempenho (lado do cliente) com alta disponibilidade: cenário com falha do servidor web da máquina virtual 2 e do servidor MySQL ”Master” da máquina virtual 1. 56
5.10 Resultados obtidos dos testes efetuados com a ferramenta do desempenho do website (lado do servidor). 58

Lista de Acrónimos

Amazon EC2	Amazon Elastic Compute Cloud
AMIs	Amazon Machine Images
API	Application Programming Interface
AWS	Amazon Web Services
CDN	Content Delivery Network
CPU	Central Process Unit
HA	High Availability
HTTP	Hypertex Transfer Protocol
IaaS	Infrastructure as a Service
IP	Internet Protocol
MPI	Message Passing Interface
NASA	National Aeronautics and Space Administration
NAT	Network Address Translation
NIST	National Institute of Standarts and Technology
NTP	Network Time Protocol
PaaS	Platform as a Service
POP	Point of Presence
RAM	Random Access Memory
REST	Representational State Transfer
SaaS	Software as a Service
SDN	Remote Procedure Call
SDN	Software-Defined Networking
SPOFs	Single Point of Failure
SSH	Secure Shell
TCP	Transmission Control Protocol
TI	Tecnologia de Informação
TIC	Tecnologia de Informação e Comunicação
TTFB	Time To First Byte
URL	Uniform Resource Locator
VM	Virtual Machine

Capítulo 1

Introdução

O foco deste trabalho é estudar os conceitos inerentes à computação na cloud, que é uma tecnologia que atualmente desempenha um papel preponderante no contexto das organizações. A computação na cloud, para além de abarcar todas as características de um sistema informático tradicional, disponibiliza às organizações/empresas vários recursos de computação sem que essas façam um investimento em infraestruturas próprias. A elas só cabe pagar pelos recursos consumidos [1] [2]. É um modelo de computação recente [1], que fornece aos utilizadores inúmeros recursos computacionais (hardware, rede, armazenamento, etc.) como um serviço, que é acedido através da internet sem que os utilizadores controlem a infraestrutura [2]. O objetivo deste trabalho é estudar os mecanismos de tolerância a falhas na cloud e implementar uma aplicação na cloud com elevada disponibilidade avaliando o seu desempenho.

Na vertente prática deste trabalho, foi instalado e configurado o sistema cloud OpenStack versão "Mitaka", foi instalada a versão básica, isto é, com três nós, Controlador (Controller), Computação (Compute) e Armazenamento em Bloco (Block Storage).Como forma de materializar os conceitos de mecanismos de tolerâncias a falhas e alta disponibilidade explanadas no Estado da Arte, foi implementado neste sistema de cloud, um mecanismo de tolerância a falhas a nível das aplicações. Para tal, desenvolveu-se uma aplicação com elevada disponibilidade utilizando as seguintes ferramentas para alta disponibilidade de software, o servidor HAProxy para o balanceamento de carga dos servidores web e a configuração da réplica dos servidores MySQL. O desempenho da aplicação na cloud foi testado com as ferramentas do desempenho de aplicação web, do lado do cliente e do lado do servidor. Os testes do desempenho da aplicação na cloud foram realizados em dois momentos distintos: sem tolerância a falhas e com tolerância a falhas (réplica dos servidores).

As contribuições deste trabalho são um estudo do estado da arte da computação na cloud; um estudo de mecanismos de tolerância a falhas em sistemas de cloud; implementação em OpenStack de uma arquitectura de alta disponibilidade para aplicações web e avaliação de desempenho da arquitectura implementada para vários cenários de falhas.

Este trabalho é constituído por seis capítulos. O capítulo I faz a introdução do trabalho, referindo-se ao objetivo e sua organização. O capítulo II apresenta os conceitos de computação na cloud, e ainda faz referência aos sistemas de computação na cloud destacando a Framework OpenStack. O capítulo III descreve os conceitos de tolerâncias a falhas com enfoque sobre os tipos de falhas, técnicas de tolerância a falhas e alta disponibilidade. O capítulo IV debruça-se sobre o ambiente experimental e ferramentas de avaliação de desempenho descrevendo a instalação da infraestrutura de cloud OpenStack, e o cenário de elevada disponibilidade implementado. O capítulo V apresenta os resultados da avaliação do desempenho da aplicação na cloud. Por último, no capítulo VI são apresentadas as principais conclusões do trabalho realizado, assim como as limitações encontradas e possíveis trabalhos futuros.

Capítulo 2

Computação na Cloud

2.1 Introdução

Este capítulo apresenta os conceitos de computação na cloud, destacando as suas características, os seus modelos de serviços e os tipos de implementação. Também se debruça sobre a descrição da framework OpenStack que é o sistema cloud implementado neste trabalho. Ainda apresenta o conceito de virtualização que é uma tecnologia utilizada na implementação da cloud e finalmente faz referências a outros sistemas da computação na cloud.

2.2 Definição e Características

Segundo Taurion [3] o termo computação na cloud, surgiu em 2006 numa palestra de Eric Schmidt, do Google, sobre como sua empresa fazia a gestão dos seus data centers. Hoje, computação na cloud, se apresenta como o cerne de um movimento de profundas transformações do mundo da tecnologia. A tarefa de definir "cloud computing" está longe de ser consensual entre os especialistas na área. Todavia, de forma cíclica têm surgido várias definições. De seguida são descritas algumas definições interessantes e relevantes encontradas na literatura.

A computação na cloud de acordo com o NIST (National Institute of Standarts and Technology) pode ser entendida como um modelo de computação que permite o acesso, de modo conveniente e a pedido, a um conjunto de recursos de computação configuráveis (como por exemplo redes, servidores, armazenamento, serviços web, aplicação e virtualização) que podem ser rapidamente aprovisionados e libertados com esforço mínimo de gestão ou interação com o provedor de serviços [4].

De acordo com os autores Madani e Jamali [5] a computação na cloud, é vista como um método de computação que fornece aos utilizadores os recursos da tecnologia da informação como um serviço, e permite que eles tenham acesso a esses serviços na Internet sem precisar de informações especializadas ou controlar a infraestrutura.

Dentro deste contexto, pode-se afirmar que a computação na cloud refere-se tanto às aplicações oferecidas como serviços pela Internet, como também ao software, hardware e sistemas nos data centers que fornecem esses serviços [6]. O objetivo primordial da Computação na cloud é transformar a indústria de Tecnologia da Informação (TI), tornando o software cada vez mais atraente como serviço e moldando a forma como o hardware é projetado e adquirido [6]. Isto porque, desta feita os desenvolvedores com ideias inovadoras para novos serviços de Internet, não necessitam de fazer grandes desembolsos de capitais em hardware para implantar o seu serviço nem para custear a despesa humana para operá-lo.

As características essências do modelo de computação na cloud [4] são:

- On-demand self (auto-atendimento sob demanda): um utilizador pode alocar recursos de computação (instâncias, armazenamento em rede etc.), conforme necessário, automaticamente, isto é, sem a intervenção do provedor do serviço;
- Broad network access (amplo acesso à rede): os recursos disponíveis na rede podem ser acedidos por meio de mecanismos padrão que promovem o uso por plataformas heterogéneas, como por exemplo, telemóveis, tablets, laptops ou estações de trabalho;
- Resource pooling (agrupamento de recursos): o provedor do serviço disponibiliza os recursos (armazenamento, processamento, memória e largura de banda de rede) existentes a diferentes utilizadores, recursos esses, que são reservados e libertados de acordo com a necessidade momentânea de cada utilizador;
- Rapid Elasticity (elasticidade rápida): os recursos podem ser provisionados e libertados de forma elástica, em alguns casos automaticamente. Ou seja, torna-se crucial que os recursos sejam elásticos para que possam ser rapidamente redimensionados consoante o utilizador necessite de ampliar, ou reduzir os mesmos recursos. Para o utilizador, os recursos disponíveis para provisionamento muitas vezes parecem ilimitados e podem ser apropriados em qualquer quantidade e a qualquer momento;
- Measured service (medição dos serviços): os sistemas da computação na cloud controlam e monitorizam automaticamente os recursos para cada tipo de serviço (armazenamento, processamento e largura de banda). Esse monitoramento do uso dos recursos deve ser transparente para o provedor de serviços, assim como para o utilizador do serviço utilizado.

2.3 Modelos de Serviços de Computação na Cloud

A computação na cloud permite aos utilizadores aceder softwares de aplicação e recursos de computação utilizando modelos de serviços [7]. Atualmente, de acordo, com a maioria das literaturas a computação na cloud é composta por três camadas [8], como mostra a figura 2.1, são elas:

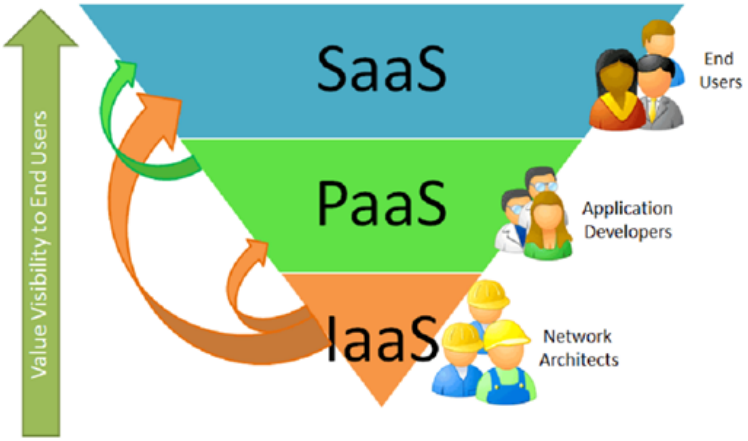

Figura 2.1: Modelos de computação na cloud [9].

- Infraestrutura como um Serviço – IaaS
- Plataforma como um Serviço – PaaS

- Software como um Serviço – SaaS

2.3.1 Infraestrutura como um Serviço-IaaS

IaaS é a camada inferior do modelo conceitual, nela o provedor tem a capacidade de fornecer ao utilizador recursos fundamentais de computação (como processamento, armazenamento, redes etc.). O consumidor pode implantar e executar diferentes tipos de software, incluindo sistemas operativos [7]. Nesta camada o utilizador não gere e nem controla a infraestrutura física todavia, através de mecanismos de virtualização, tem controlo sobre os sistemas operativos, armazenamento, aplicações instaladas e um controlo limitado dos componentes da rede (como por exemplo, host e firewall) [8].

2.3.2 Plataforma como um Serviço-PaaS

PaaS pode ser entendida como a camada intermediária do modelo conceitual, sendo composta por hardware virtual disponibilizado como serviço. Este modelo permite ao utilizador implementar aplicações criadas por ele mesmo numa plataforma de cloud virtualizada. O modelo PaaS inclui middleware, base de dados, ferramentas de desenvolvimento e algum suporte de execução, como a Web 2.0 e o Java. A plataforma inclui hardware e software integrado com interfaces de programação específicas. O provedor fornece a API (Application Programming Interface) e ferramentas de software (por exemplo, Java, Python, Web 2.0, NET) [10].

Nesta camada também o utilizador não gere e nem controla a infraestrutura (rede, servidores, sistemas operativos ou armazenamento), mas tem controlo sobre as aplicações instaladas e definições de configuração para a hospedagem de ambiente de aplicações [8].

2.3.3 Software como um Serviço-SaaS

SaaS é a camada mais externa do modelo conceitual. Neste modelo, o utilizador beneficia da capacidade de utilizar aplicações já implantados no ambiente de cloud por um provedor [7]. Isto é, as aplicações estão hospedadas na cloud e podem ser acedidas por utilizadores para as mais diversas finalidades. Os softwares são acedidos a partir de vários dispositivos, como por exemplo smartphone, smartwatchs e smart tvs. De forma similar aos outros dois modelos referenciados, o utilizador não gere nem controla a infraestrutura subjacente, nem as aplicações individuais, com a possível exceção das suas configurações específicas [8].

2.4 Tipos de Implementação da Cloud

Nem todas as clouds são iguais. Há quem defenda que existem três formas diferentes de implementar os recursos de informática na cloud : cloud pública, cloud privada e cloud híbrida [11]. Outros autores consideram mais um de tipo de implementação que é cloud comunitária [2] [8]. De seguida descrevem-se estes quatro modelos de implementação de Cloud.

2.4.1 Cloud Privada

No modelo de implementação de cloud privada, a infraestrutura de cloud é construída dentro do domínio duma intranet e é utilizada exclusivamente para uma organização [10]. Ela é admi-

nistrada pela própria empresa ou por terceiros, ou por ambas as partes, e pode ser implantada dentro ou fora das instalações da organização [8].

As cloud privadas fornecem aos utilizadores locais uma infraestrutura privada flexível e ágil para executar cargas de trabalho de serviço dentro de seus domínios administrativos. O acesso aos serviços é limitado aos clientes proprietários e os seus parceiros, isto é, são aplicadas as políticas de autenticação e autorização para aceder aos serviços [10]. As cloud privadas são construídas exclusivamente para um único utilizador (uma empresa, por exemplo), com o intuito de maximizar e otimizar os recursos internos existentes, também por motivos de segurança e confiança nas cloud privadas [1].

2.4.2 Cloud Pública

Numa cloud pública, a infraestrutura pertence a uma provedor que vende serviços para o público em geral e pode ser acedido por qualquer utilizador que conheça a localização do serviço, desde que tenha pago pelo serviço usufruído [10]. Nas clouds públicas, o hardware, o software e as outras infraestruturas de apoio são detidas e geridas pelo fornecedor de serviços cloud. O acesso a estes serviços e a gestão da conta são feitos através da conexão online [1].

2.4.3 Cloud Híbrida

As clouds híbridas podem ser entendidas como a combinação das clouds pública e privada, vinculadas através de uma tecnologia padronizada que permite que os dados e as aplicações sejam partilhados entre elas [1].

O modelo de cloud híbrida possibilita manter sistemas na cloud privada e outros na cloud pública, simultaneamente. Por exemplo, sistemas críticos ou que manipulam informações confidenciais podem ser hospedados internamente enquanto outros sistemas, que não lidam com dados sigilosos, podem ser utilizados em uma rede pública [11].

As cloud públicas promovem a padronização, preservam o investimento de capital e oferecem flexibilidade de aplicação, enquanto que as cloud privadas tentam obter personalização e oferecem maior eficiência, resiliência, segurança e privacidade [10].

2.4.4 Cloud Comunitária

A infraestrutura da cloud pode ser compartilhada por um conjunto de empresas que partilhem os mesmos interesses, características ou preocupações (como por exemplo nas políticas de autenticação, requisitos de segurança ou em uma missão), podendo assim partilhar os custos entre elas. Nesta infraestrutura da cloud a administração é feita por uma ou mais empresas participantes da comunidade ou por uma empresa terceira ou ainda por uma combinação entre elas [8].

2.5 A Framework OpenStack

O OpenStack é uma framework “open source” que vem ganhando espaço na comunidade de software livre e na indústria [12]. O OpenStack fornece software para construir uma IaaS, foi

iniciado num projeto em 2010, através da colaboração entre as empresas Rackspace e a agência espacial americana, a NASA [11] [13]. Atualmente, especialistas de computação na cloud de todo o mundo contribuem para o projeto. O OpenStack desenvolve projetos como o OpenStack Compute, que oferece recursos de computação para máquinas virtuais e gestão de rede, e OpenStack Object Storage, que proporciona um serviço de armazenamento de objetos escaláveis [12].

A framework OpenStack inclui vários serviços que são instalados separadamente. Esses serviços funcionam juntos, dependendo das necessidades da cloud. São eles: serviços de computação (compute), identidade (keystone), rede (neutron), imagem (glance), armazenamento em bloco (cinder), armazenamento de objetos (swift), serviços de telemetria (Ceilometer), base de dados, horizon (dashboard). Ainda é de realçar que qualquer um desses serviços pode ser instalado e configurado separadamente ou como entidades conectadas.

2.5.1 Arquitectura dos Componentes do Openstack

A arquitectura da plataforma OpenStack depende da sua versão, neste trabalho a versão utilizada é Openstack ”Mitaka”.

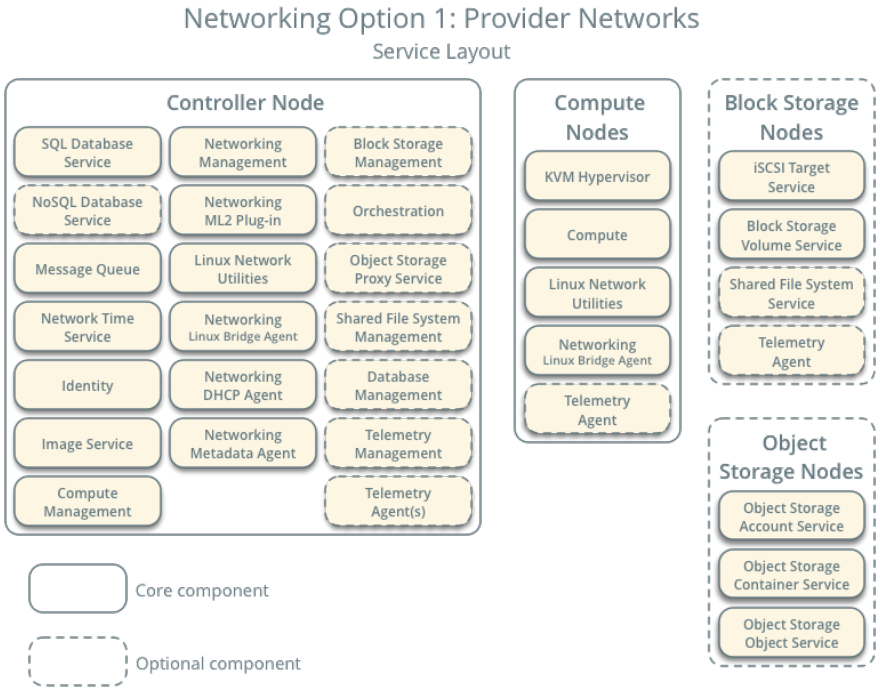

Figura 2.2: Arquitectura dos componentes do Openstack [14].

A versão Openstack ”Mitaka” requer pelo menos dois nós (hosts) conforme a figura 2.2, para iniciar uma VM (Virtual Machine) ou instância básica, esses nós básicos são: “Controller” e “Compute” . Os outros nós podem ser serviços opcionais, como Armazenamento de Blocos (“Block Storage”) e Armazenamento de Objetos (“Object Storage”) [14]. Todavia, se a infraestrutura for implementada sem o nó Armazenamento de Blocos (“Block Storage”), não vai ser possível anexar volumes às máquinas virtuais instanciadas, porque é este nó que disponibiliza o volume de armazenamento. Sendo assim, sem o nó Armazenamento de Blocos (“Block Storage”) as

máquinas virtuais instanciadas só terão o sistema operativo versão experimental, logo ao serem encerradas (VMs) todos os dados nelas armazenadas serão perdidos, visto que elas (VMs) não tem volume de armazenamento (só têm um disco efémero). Ainda é de realçar, que o drive de volume instalado no Block Storage é o drive LVM, isto porque este drive suporta o sistema Operativo Linux versão Ubuntu 14.04 LTS.Este sistema operativo foi instalado nas máquinas que contêm os nós da infraestrutura de cloud implementada neste trabalho e nas máquinas virtuais nele instanciadas.

Controlador (Controller)

O nó "Controller" executa o serviço de keystone (identidade), o serviço de glance (imagem), e alguns serviços do Compute. Também faz gestão de vários agentes de rede e o Horizon(dashboard). Ele também inclui serviços de suporte, como base de dados SQL, fila de mensagens e NTP (Network Time Protocol) [14]. Opcionalmente, o nó "Controller" executa alguns dos serviços dos nós Armazenamento em Bloco e Armazenamento de Objetos, Orquestração e Telemetria. O nó "Controller" requer no mínimo duas interfaces de rede [14].

Computação (Compute)

O nó "Compute" executa a parte do hypervisor do Compute para gestão de máquinas virtuais. Pode-se implantar mais de um nó "Compute". Cada nó requer no mínimo duas interfaces de rede. Por padrão, o Compute utiliza o hypervisor KVM . O nó Compute também executa um agente de serviço de rede que conecta instâncias a redes virtuais e fornece serviços de firewall a instâncias por meio de grupos de segurança [14].

Armazenamento em bloco (Block storage)

O nó opcional "Block Storage" contém os discos que os serviços de Block Storage e sistema de ficheiros compartilhado provisionam para instâncias [14]. O tráfego de serviço entre os nós "Compute" e "Block Storage" é feito através do módulo "Network Management". Pode-se implantar mais do que um nó de "Block Storage". Cada nó requer no mínimo uma interface de rede [14].

2.5.2 Serviços do Openstack

Como foi dito anteriormente, o projeto OpenStack é uma framework de computação de cloud de código "open source" constituída por diversos serviços. Nessa subsecção serão descritos os serviços mais importantes de OpenStack versão "Mitaka". A escolha recaiu sobre a versão "Mitaka" porque na altura da realização do trabalho, era a versão mais estável e também apresenta uma arquitectura que consume poucos recursos computacionais, isto é, requer no mínimo dois nós. Os principais serviços são [14]:

Ceilometer (Serviço de telemetria)

O Ceilometer (serviço de telemetria) é um serviço que faz a monitorização e métrica de recursos para o Openstack.

Cinder (Serviço de armazenamento em bloco)

O Cinder (serviço de armazenamento em bloco) fornece o armazenamento baseado em bloco persistente para instâncias (máquinas virtuais criadas na infraestruturas da cloud OpenStack) em execução. O método sobre a qual o armazenamento é aprovisionado e consumido é determinado pelo driver de armazenamento em bloco ou pelos drivers no caso de uma configuração de vários back-ends. Há uma variedade de drivers disponíveis: NAS / SAN, NFS, LVM, Ceph, e muito mais. Neste trabalho foi instalado o driver LVM (Logical Volume Manager) e a conexão com as máquinas virtuais é feita através do protocolo iSCSI(Internal Small Computer System Interface). O iSCSI é usado pra facilitar a transferência de dados pelas intranets e para gerir armazenamento de longas distâncias [1], neste caso, entre o Block Storage e as máquinas virtuais.

Glance (Serviço de imagem)

O Glance (serviço de imagem) armazena e recupera imagens de disco da máquina virtual. O OpenStack Compute faz uso do glance durante o provisionamento da instância. Isto é, imagens de disco permitem que uma instância possa ser criada sem perda de tempo com instalação de sistema operativo e configuração de ambiente. Glance permite a criação e registo de imagens pré-configuradas que podem servir como base para a criação de novas instâncias de máquinas virtuais. Ele oferece uma API REST (Representational State Transfer) que permite consultar metadados de imagens de máquinas virtuais e recuperar uma imagem real.

Horizon (Serviço de dashboard)

O Horizon (serviço de dashboard) fornece um portal de auto-atendimento baseado na Web para interagir com os serviços subjacentes do OpenStack, como o lançamento de uma instância, a atribuição de endereços IP e a configuração de controlo de acesso. Isto é, é uma interface da Web que permite aos administradores e utilizadores da cloud gerir vários recursos e serviços (Nova, Neutron, Swift, Glance, etc) do OpenStack, agilizando e facilitando o processo da gestão da Cloud.

Keystone (Serviço de identidade)

O serviço keystone é responsável pela autenticação, gestão de identidades e projetos, e autorização para aceder a outros serviços do OpenStack. O keystone fornece um ponto único de integração para gerir serviços de autenticação, autorização e catálogo de serviços. Ou seja, cada um dos serviços do OpenStack é registado no Keystone. Cada um dos serviços possui um endpoint [2] e um utilizador. O serviço keystone desempenha um papel fundamental para os outros serviços que utilizam o serviço de keystone como uma API unificada comum do OpenStack. É ele que sustenta a verificação e listagem de serviços disponíveis a todos os outros serviços do OpenStack. Quando um serviço do OpenStack recebe uma solicitação de um utilizador, ele verifica com o serviço de keystone se o utilizador está autorizado a fazer tal solicitação.

Neutron (Serviço de rede)

[1] https://pt.wikipedia.org/wiki/iSCSI

[2] São URLs utilizadas para aceder esses serviços

O Neutron (serviço de rede) é o serviço de rede do OpenStack que ativa a conectividade de rede como serviço para outros serviços do OpenStack, como por exemplo o OpenStack Compute. Fornece uma API aos utilizadores para definirem as suas próprias redes e associarem interfaces do servidor a ela. A sua arquitectura baseia-se em plugins que permite aos utilizadores usufruir das vantagens de recursos disponíveis nos dispositivos de rede específicos. Isto é, os plug-ins podem ser implementados para acomodar diferentes equipamentos de rede e software, fornecendo flexibilidade para a arquitectura e implementação do OpenStack.

Qualquer rede configurada deve ter, pelo menos, uma rede externa. Ao contrário das outras redes, a rede externa não é apenas uma rede virtual. Representa uma abstração da rede física, sendo acessível por qualquer endereço IP fora do ambiente virtual que esteja ligado diretamente à rede física ou que possua rota definida.

Além de redes externas, qualquer rede configurada tem uma ou mais redes internas (sub-redes). Estas redes definidas por software conectam-se diretamente às máquinas virtuais (Virtual Machines-VMs). Apenas as VMs da rede interna, ou aquelas em sub-redes conectadas através de um router, podem aceder VMs conectados diretamente a essa rede.

Os routers virtuais são necessários para conectar as VMs à rede exterior. Cada router tem uma porta de entrada ligada a uma rede externa e uma ou mais interfaces ligadas a redes internas. Assim como um router físico, sub-redes podem aceder máquinas em outras sub-redes que estão conectados ao mesmo router, e as máquinas podem aceder à rede externa através deste router. Para que uma VM seja acedida externamente é necessário alocar um endereço IP externo, ou endereço público.

Nova (Serviço de computação)

O serviço “nova” gere o ciclo de vida de instâncias de computação no ambiente do OpenStack, ou seja permite iniciar e parar instâncias virtuais, é responsável pela gestão da rede virtual para cada instância que faz parte dum projeto. Também permite criar e editar grupos de segurança e adicionar ou remover volumes. Para o seu funcionamento requer os seguintes serviços básicos do OpenStack: keystone, glance e neutron. Por exemplo, um utilizador autenticado com acesso ao glance e a uma rede configurada para as instâncias de máquinas virtuais no OpenStack já tem todas as condições para criar novas máquinas virtuais. Os requisitos necessários são um par de chaves[3] e um grupo de segurança.

O serviço “nova” é composto pelos seguintes componentes:

- Nova-api: aceita e responde a chamadas de API de computação do utilizador final;
- Nova-conductor: faz a mediação das interações entre o serviço nova-compute e a base de dados;
- Nova-compute: cria e encerra instâncias de máquina virtual por meio de APIs do hypervisor;
- Nova-scheduler: serviço que determina como responder às solicitações de computação;

[3]O par de chaves utilizado pelo OpenStack é simplesmente um par de chaves SSH (Secure Shell)

- Base de dados: efetua o armazenamento dos estados (VMs em execução, redes disponíveis) "build-time" e "run-time" da infraestrutura cloud.

Swift (Serviço de armazenamento de objetos)

O Swift (serviço de armazenamento de objetos) é o serviço responsável pelo armazenamento e recuperação de objetos através de uma API baseada em HTTP RESTful. É escalável e pode gerir grandes quantidades de dados não estruturados a baixo custo. A autenticação e autorização para leitura e escrita de dados é gerida pelo Keystone. O Swift armazena conteúdo de dados, como documentos, imagens, backups, vídeos e assim por diante.

2.6 Outras Sistemas de Computação na Cloud

Esta secção descreve outros sistemas de computação na cloud. Isto é, faz uma breve descrição de outros provedores de serviços de computação na cloud.

2.6.1 Amazon Web Services (AWS)

A Amazon Web Services (AWS) é uma plataforma de serviços na cloud que oferece um conjunto de serviços, nomeadamente computação, armazenamento, base de dados, distribuição de conteúdo e outras funcionalidades para ajudar as empresas no seu dimensionamento e crescimento [15]. Alguns dos serviços da computação da AWS são: Amazon EC2 (Elastic Compute Cloud), AWS Lambda.

O Amazon EC2 é um serviço web que oferece capacidade de computação redimensionável na cloud. Isto é, o Amazon EC2 pode ser entendido como um ambiente de computação virtual, permitindo aos utilizadores, através de uma interface Web, criar, utilizar e gerir máquinas virtuais com sistemas operativos Windows e Linux, ou mesmo iniciar tais máquinas de acordo com as necessidades das aplicações. De acordo com as regras da Computação na cloud, o utilizador paga apenas pelos recursos consumidos, por instâncias e/ou transferência de dados (cobrado por gigabyte de dados transferidos) [16].

O Amazon EC2 fornece uma ferramenta denominada AMIs (Amazon Machine Images), que funcionam como uma espécie de template e contém uma pré-configuração de software (por exemplo, sistema operativos e aplicações), a partir das quais se podem criar instâncias (máquinas virtuais), que são cópias executáveis da AMI. Essas instâncias, que podem ser múltiplas e inclusive de diferentes tipos, são executadas até que sejam paradas ou finalizadas pelo utilizador; se uma instância falhar por uma razão ou outra, pode-se criar uma nova a partir da AMI seleccionada [16].

O AWS Lambda também é um serviço de computação da AWS, que permite executar todas as aplicações para back-end sem ser necessário possuir ou gerir servidores. Isto é, basta executar a aplicação, o AWS Lambda encarrega-se de todos os itens necessários para executá-lo e alterar a sua escala com alta disponibilidade se for necessário. Neste serviço o utilizador paga consoante o tempo que utilizou os recursos [17].

No que tange ao armazenamento da AWS, um dos serviços da AWS é o Amazon Elastic Block Store

(EBS), que é um sistema de armazenamento em bloco usado para armazenar dados persistentes. O Amazon EBS é adequado para instâncias do EC2, fornecendo volumes de armazenamento de nível de bloco. Ele possui três tipos de volumes que são: "General Purpose" (SSD), "Provisioned IOPS" (SSD), e "Magnetic". Esses três tipos de volume diferem em desempenho, características e custo [18].

2.6.2 Microsoft Windows Azure Plataform

O Windows Azure é fornecido pela Microsoft como uma plataforma como um serviço (PaaS). Esta plataforma fornece serviços acessíveis ao desenvolvedor para criar aplicações e armazenar dados. Os seus principais elementos são: Windows Azure, SQL Azure e Windows Azure AppFabric [19].

Windows azure

O Windows Azure é composto por cinco componentes [19]:

- Compute: o Windows Azure Compute fornece aos desenvolvedores uma plataforma para execução, armazenamento e gestão de aplicações;
- Armazenamento: o serviço de armazenamento permite armazenar recursos na cloud. Esses recursos podem ser ficheiros, como documentos, imagens ou vídeos, juntamente com informações relevantes de metadados, ou podem ser informação estruturada ou semiestruturada;
- "Fabric controller": a principal tarefa de Fabric Controller é fornecer uma visão única, sobre a qual os serviços de computação e armazenamento estão localizados;
- "Content Delivery Network"- (CDN) [20]: as redes de entrega de conteúdos são redes distribuídas de servidores que podem entregar, de forma eficiente, conteúdos da Web aos utilizadores. As CDNs armazenam os conteúdos em cache em servidores Edge em localizações do ponto de presença (POP) que estão próximas dos utilizadores finais, para minimizar a latência;
- "Connect": permite aceder aplicações na cloud duma forma segura como se o cliente estivesse dentro da intranet da organização. Também permite criar uma conexão direta entre a plataforma baseada na cloud e a organização da data center.

SQL azure

O elemento SQL Azure da plataforma Azure fornece serviços de armazenamento para bases de dados relacionais.

Windows azure appFabric

O Windows Azure AppFabric funciona como uma camada de middleware baseada na cloud, como forma de integrar aplicações existentes, sendo muito útil em situações de cloud híbridas [19].

2.6.3 Aneka

A Aneka é uma plataforma de computação na cloud desenvolvida pela Manjrasoft, em Melbourne, Austrália. Ela é projetada para suportar o desenvolvimento e implantação de aplicações distri-

buídas em cloud. Isto é, desempenha o papel de PaaS (Plataforma como um Serviço) para a computação na cloud [21]. Ela (Aneka) é composta por uma coleção de recursos físicos e virtuais conectados através de uma rede, que pode ser a Internet ou uma intranet privada. Cada um desses recursos hospeda uma instância do Aneka Container representando o ambiente de execução onde as aplicações distribuídas são executadas [21].

2.7 Virtualização

A virtualização na computação é abstração dos componentes físicos em objetos computacionais lógicos [22]. Isto é, os recursos computacionais reias são "transformados" em recursos virtuais, ou seja consiste na criação de um ambiente virtual que simula um ambiente real permitindo o uso de sistemas operativos e aplicações [23]. Pode-se virtualizar uma plataforma de hardware (processadores, memória, armazenamento e conectividade de rede), ou um sistema operativo [24]. A forma de virtualização mais conhecida é através de máquina virtual ou "Virtual Machine (VM) ". Uma máquina virtual é a simulação de um computador real, ela oferece um ambiente completo, muito semelhante ao de uma máquina física real, tendo o seu próprio sistema operativo, softwares de aplicação e serviços de rede. De uma outra forma, criar uma máquina virtual, é a mesma coisa que criar um computador dentro de outro computador. É executada numa janela do Computador real, tal como qualquer outro programa, dando ao utilizador final a mesma experiência que teria se utilizasse o próprio sistema operativo anfitrião [25]. Para criar uma máquina virtual, antes precisa de se instalar um software que fornece um ambiente sobre o qual as máquinas virtuais se operam. Este software é chamado VMM (Virtual Machine Monitor - Monitor de Máquina Virtual). O VMM é um componente de software que hospeda as máquinas virtuais, também chamado de hypervisor [22].Existem múltiplas soluções de virtualização [26], isto é, vários VMMs ou hypervisor, como por exemplo: VMWare, KVM, Xen, QEMU etc. Neste trabalho, o hypervisor utilizado para a virtualização foi KVM (Kernel-based Virtual Machine), visto que é o recomendado pela Infraestrutura Cloud OpenStack Versão "Mitaka".
A computação na cloud utiliza a virtualização de recursos computacionais disponíveis como por exemplo: máquinas virtuais, armazenamento virtual e redes virtuais. No caso de serviços de computação e da utilização de máquinas virtuais, o backup é um método que não pode faltar em sistemas virtualizados, visto que a imagem virtual contém tudo o que é necessário para executar a aplicação e pode ser transparente para migração entre máquinas físicas [27].

A virtualização oferece inúmeras vantagens, entre as quais se destacam [28]:

- Melhor aproveitamento da infraestrutura existente: ao executar vários serviços num servidor ou conjunto de máquinas, por exemplo, pode-se aproveitar a capacidade de processamento destes equipamentos o mais próximo possível de sua totalidade;
- O parque de máquinas é menor: com o melhor aproveitamento dos recursos já existentes, a necessidade de aquisição de novos equipamentos diminui, assim como os consequentes gastos com instalação, espaço físico, refrigeração, manutenção, consumo de energia, entre outros;
- Gestão centralizada: dependendo da solução de virtualização utilizada, fica mais fácil monitorar os serviços em execução, já que a sua gestão é feita de forma centralizada;
- Diversidade de plataformas: pode-se ter uma grande diversidade de plataformas e, assim, realizar testes de desempenho de determinada aplicação em cada uma delas, por exemplo.

Todavia, ainda é de realçar que a virtualização também tem as suas desvantagens, nomeadamente são [28]:

- Sobrecarga das máquinas virtuais: a quantidade de máquinas virtuais que um computador pode suportar não é ilimitada, razão pela qual é necessário encontrar um equilíbrio para evitar sobrecarga, caso contrário, o desempenho de todas as máquinas virtuais será afetado;

- Segurança: se houver uma vulnerabilidade de segurança no VMM, por exemplo, todas as máquinas virtuais poderão ser afetadas pelo mesmo problema;

- Contingência: nas aplicações críticas, é importante ter um computador que possa atuar imediatamente no lugar da máquina principal (como um servidor), pois se esta parar de funcionar, todos os sistemas virtualizados que rodam nela também serão interrompidos;

- Desempenho: a virtualização pode não ter bom desempenho em todas as aplicações, por isso é importante avaliar muito bem a solução antes de sua efetiva implementação.

2.8 Trabalhos Relacionados

Alguns trabalhos já foram desenvolvidos na área de tolerância a falhas e alta disponibilidade na infraestrutura de cloud OpenStack e nos servidores nele alojado.

Um dos exemplos é o trabalho de [MARTINS,2014] [27] intitulado de **"Tolerância a Falha em um Ambiente de Computação em Nuvem open source"**. Na sua obra ele desenvolveu um mecanismo tolerante a falhas no OpenStack. Implementou um mecanismo de redundância nas máquinas virtuais instanciadas nos nós da cloud, em que se um nó apresentar uma falha transitória ou intermitente, a máquina virtual ficará armazenada num local seguro, aguardando que o nó recupere da falha. Após a implementação concluiu que o mecanismo desenvolvido é viável e eficiente, isto porque, quando o nó se recuperar duma falha, a máquina virtual nela instanciada não é perdida, voltando a ficar ativa e disponível para o utilizador.

No trabalho de [EMER,2016] [29] com título de **"Implementação de alta disponibilidade em uma empresa prestadora de serviços para Internet"**, foi implementado um ambiente de alta disponibilidade para uma empresa prestadora de serviços para internet utilizando ferramentas de código aberto. Desenvolveu-se uma solução de alta disponibilidade, baseada no uso de um "Cluster" de computadores e virtualização. Para efetuar a implementação foi necessário um software para a replicação e outro para a gestão do "Cluster" das máquinas virtuais. Para a gestão do "Cluster" das máquinas virtuais foi utilizado o software Pacemaker e para a replicação dos dados foi utilizado o software DRBD (Distributed Replicated Block Device). Após a implementação foram executados testes para simular falhas de hardware, de energia elétrica e de software, com o objetivo de validar o ambiente de alta disponibilidade desenvolvida. Por fim chegou se a conclusão que o tempo de indisponibilidade dos serviços no ambiente de alta disponibilidade é consideravelmente menor em relação ao antigo ambiente da empresa (sem alta disponibilidade).

2.9 Conclusão

Este capítulo serviu de base para clarificar os conceitos associados a computação na cloud. Também contribuiu para compreender o funcionamento da infraestrutura da cloud OpenStack, isto é, o OpenStack é framework "open source" que fornece uma solução de infraestrutura como um serviço. A arquitectura do framework OpenStack depende da sua versão, em cada seis meses é lançada uma versão. Neste trabalho foi implementado a versão OpenStack "Mitaka". Ainda de realçar que existem outros sistemas de cloud, como por exemplo o Amazon Web Services, Microsoft Windows Azure Plataform, Aneka etc. Por fim, concluiu que a tecnologia de virtualização está associado a computação na cloud, pois, permite criar máquinas virtuais e outros tipos de virtualização como por exemplo virtualizar processadores, memória etc.

Capítulo 3

Tolerância a Falhas

3.1 Introdução

Este capítulo apresenta os conceitos de tolerância a falhas, listando quais os tipos de falhas e técnicas de tolerância a falhas. Para além dos conceitos de tolerância a falhas, abarca os conceitos de alta disponibilidade, com incidência sobre a alta disponibilidade na infraestrutura da cloud OpenStack, mencionando as tecnologias que permitem a alta disponibilidade através da redundância de hardware, e as ferramentas de alta disponibilidade no software.

3.2 Conceitos

Na tentativa de implementar sistemas mais confiáveis, foram desenvolvidos meios para oferecer mais confiança aos sistemas, este processo chama-se tolerância a falhas. Sabendo que falhas são inevitáveis, procura-se atribuir aos sistemas a capacidade de tolerar a ocorrência de falhas, apresentando funcionamento desejado ou pré-definido, evitando assim danos ao utilizador [30]. A tolerância a falhas pode ser entendida como a capacidade de um sistema continuar a fornecer o seu serviço mesmo na presença de avarias de alguns dos seus componentes [5] [31]. Isto é, são técnicas que fazem com que o sistema continue a funcionar de uma forma satisfatória mesmo na presença de falhas [32].O objetivo primordial de tolerância a falhas é garantir a disponibilidade e a confiabilidade dos serviços, bem como execução das aplicações. Para minimizar o impacto da falha na execução do sistema e das aplicações, as falhas devem ser antecipadas e tratadas proactivamente [33].

No contexto de tolerância a falhas, os termos fault (falha), error (erro), failure (avaria) apresentam diferentes significados.

Falha ("fault"): uma falha é uma alteração do funcionamento de um componente (hardware ou software) do sistema [34]. Uma falha pode ocorrer de forma acidental ou intencional. Podemos classificar as falhas em três grandes grupos [34]:

- Falhas de projeto (hardware ou software): uma falha pode ocorrer em qualquer etapa do desenvolvimento de um sistema: especificação, desenho, implementação;
- Falhas físicas: defeitos de produção, deterioração dos componentes, interferência;
- Falhas de interação homem – máquina: inputs errados, ataques ou intrusões.

Erro ("error"): um erro é a manifestação de uma falha [34]. Um erro provoca a corrupção de elementos de dados (afecta o estado do sistema). Se um erro causa ou não a avaria do sistema depende de [34]: composição do sistema (por exemplo, existe redundância que mascare a ocorrência do erro) e atividade do sistema (por exemplo, o estado que contém o erro pode

não ser suficientemente duradoiro para causar uma avaria).

Avaria ("failure"): uma avaria é qualquer alteração do comportamento do sistema em relação ao esperado (i.é, em relação à sua especificação) [34]. A figura 3.1 demonstra o percurso para a geração de uma avaria.

Figura 3.1: Percurso para avaria [35].

Os atributos de confiança no funcionamento de um sistema são [34]: fiabilidade (reliability), disponibilidade (availability), segurança contra falhas acidentais (safety),confidencialidade (confidentiality), integridade (integrity) e facilidade de manutenção (maintainability).

- Fiabilidade: probabilidade de o sistema funcionar de acordo com as especificações, dentro de certas condições, durante um certo período de tempo [34];
- Disponibilidade: probabilidade de o sistema estar operacional num dado instante de tempo [34];
- Segurança contra falhas acidentais: probabilidade de o sistema ou estar operacional executando as suas funções corretamente, ou parar as suas funções de forma a não provocar dano a outros sistemas ou pessoas que dele dependam [34];
- Confidencialidade: inexistência de acessos não autorizados à informação [34];
- Integridade: inexistência de alterações incorretas do estado do sistema [34];
- Facilidade de manutenção: probabilidade de um sistema com avarias ser reparado continuando a funcionar [34].

Latência

A latência de falha pode ser entendida como o período de tempo desde a ocorrência da falha até a manifestação do erro devido àquela falha. Por sua vez, a latência de erro é definido como o período de tempo desde a ocorrência do erro até a manifestação da avaria devida àquele erro. Pode se afirmar que o tempo total desde a ocorrência da falha até o aparecimento do defeito é a soma da latência de falhas e da latência de erro [36].

Em suma, num sistema de computação as falhas são inevitáveis, porque os componentes físicos envelhecem e sofrem com interferências externas, sejam ambientais ou humanas. O software, e também os projetos de software e hardware, são vítimas de sua alta complexidade e da fragilidade humana em trabalhar com grande volume de detalhes ou com deficiências de especificação.

Nó: é uma máquina (física ou virtual) que executa as operações do servidor independentemente do seu próprio sistema operativo. Como qualquer nó pode falhar, atender aos objetivos de disponibilidade requer que vários nós operem como parte de um cluster [37].

“Cluster”: são dois ou mais nós interligados por uma rede com o objetivo de aumentar o desempenho ou disponibilidade de um serviço, isto é executam os serviços em coordenação uns com os outros para completar tarefas individuais como parte de um serviço maior, onde o reconhecimento mútuo permite que um ou mais nós compensem a perda de outro [37].

“Server failure”: é a incapacidade de um nó (servidor) responder adequadamente às solicitações do cliente. Isso pode ser devido a uma falha completa, problemas de conectividade ou porque o servidor foi sobrecarregada devido a uma elevada solicitação [37].

“Failover”: é um processo no qual um outro servidor recebe os serviços que estavam a ser executados no servidor que falhou, isto é, quando um servidor falha o serviço é redirecionado para outo servidor para preencher a lacuna de serviço [37].

“Failback”: é a restauração de responsabilidades para um nó do servidor conforme ele se recupera de uma falha. Isto é, tem-se um retorno dos serviços para o servidor de origem quando este estiver disponível [37].

Replicação: é a criação de cópias de armazenamentos de dados ou processos críticos para permitir acesso síncrono confiável entre os servidores que pertencem ao “Cluster” com o intuito de garantir a redundância, o que torna o sistema tolerante a falhas, ou seja se um servidor falhar tem um outro servidor para retomar o serviço, contribuindo assim para um maior balanceamento de carga e a alta disponibilidade do sistema [37].

“Quorum” : o Quorum especifica o número mínimo de nós que devem ser funcionais num “Cluster” de nós redundantes para que o “Cluster” permaneça funcional. Quando um nó falha e o “Failover” transfere o serviço para outros nós, o sistema deve garantir que os dados e os processos permaneçam sãos. Para determinar isso, os conteúdos dos nós restantes são comparados e, se houver discrepâncias, um algoritmo de regras maioritárias é implementado [38].

3.3 Tipos de Falhas

Quando um sistema apresenta algum tipo de falha, é sinónimo dizer que o sistema não está a proporcionar os serviços de acordo com aquilo que foi preconizado. Todavia, um sistema pode apresentar algumas falhas mais ainda assim funcionar normalmente. Portanto, de acordo com a natureza da falha, deve ser realizada a ação apropriada [5]. As falhas detetadas podem ocorrer em diferentes componentes do sistema [39] [40]:

- Falha de rede: esta falha ocorre quando os dados não chegam ao destino por vários motivos, como perda de pacotes, falha de destino, falha de link e assim por diante;
- Falha física: falhas que ocorrem no hardware, como falhas da CPU, falhas de memória e assim por diante;
- Falha de mídia: erros que surgem devido à falta de meios de comunicação;
- Falha de processo: falhas proporcionadas pela falta de recursos ou erros de software;

- Falha de finalização de serviço: ocorre quando a vida útil do recurso se acaba, mas a aplicação ainda requer o uso de recursos.

As falhas podem ser divididas em várias categorias [39] [40]:

- Falha transitória: este tipo de falha ocorre uma vez e depois desaparece. Caso a operação for repetida, a falha não acontecerá novamente. Se a temporização de uma transmissão se esgotar e for executada novamente, provavelmente funcionará desta segunda vez [27];
- Falha intermitente: são falhas que se repetem de forma alternada. Estas falhas não são boas, porque resultam devido à falha de cada componente ou do funcionamento inapropriado entre os componentes, por exemplo, uma conexão defeituosa;
- Falha permanente: este tipo de falha ainda permanece no sistema,até os sistemas defeituosos serem reparados.

3.4 Técnicas de Tolerância a Falhas

Dependendo das políticas e procedimentos, as técnicas de tolerância a falhas podem ser divididas em seguintes categorias [8] [33] : tolerância a falhas reativa , tolerância a falhas proactiva e mascaramento de falhas.

Tolerância a falhas reativa

As políticas de tolerância a falhas reativas reduzem o efeito de falhas na execução da aplicação quando a falha realmente se manifesta. Existem várias técnicas baseadas nessas políticas, como "checkpoint", replicação, migração do trabalho [33] [35].

- Checkpoint: quando uma tarefa falha, é permitido que ela seja reiniciada a partir do seu estado crítico recentemente armazenado e não desde o início. Esta técnica de tolerância a falhas é eficiente no nível de tarefa para aplicações de longa duração [41];
- Replicação: várias réplicas de tarefas são executadas com recursos diferentes, para que a execução possa ter sucesso. A técnica de replicação consiste em facilitar a criação de réplicas ou cópias de um mesmo objeto em meios físicos diferentes, garantindo assim que, no caso de uma falha num dos dispositivos, o cliente seja redirecionado de forma transparente para outro que tenha uma réplica do objeto ativo [30];
- Migração de trabalho: Quando haja falha de qualquer tarefa, a tarefa em causa pode ser migrada para outra máquina. Essa técnica pode ser implementada utilizando o HAProxy para redirecionar os pedidos para outra máquina[20] [35].

Tolerância a falhas proactiva

O objetivo primordial das políticas proactivas de tolerância a falhas é evitar a necessidade de recuperação de faltas, erros e falhas, prevendo-os e proactivamente substituir os componentes suspeitos e trabalhar na manutenção de outros componentes, conforme é ilustrado na figura 3.2. Algumas das técnicas que se baseiam nessas políticas são: rejuvenescimento do software, migração preventiva, etc. [33] [35].

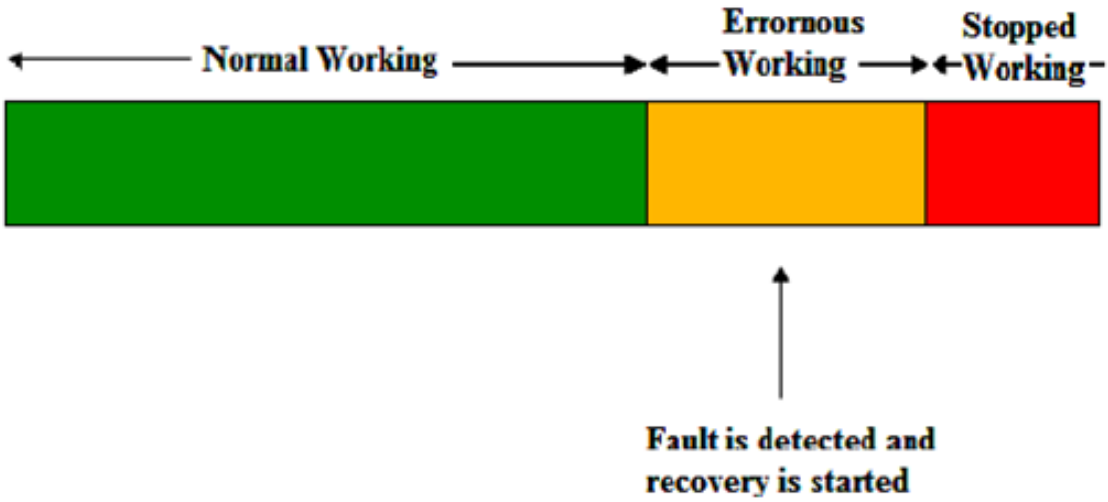

Figura 3.2: Cronograma para a precaução de um sistema de detecção de falhas [39].

- Rejuvenescimento de software: é uma técnica que projeta o sistema para reinicializações periódicas. Isto reinicia o sistema com o estado limpo [6];

- Migração preventiva: a migração preventiva depende de um mecanismo de controlo de "feedback-loop" onde a aplicação é constantemente monitorizada e analisada.

Mascaramento de falhas

O mascaramento de falhas é uma técnica de redundância estrutural que mascara as falhas dentro dum conjunto de módulos redundantes. Vários módulos idênticos executam as mesmas funções, e suas saídas são votadas para remover erros criados por um módulo defeituoso [8]. Esta técnica garante resposta correta mesmo na presença de falhas. A falha não se manifesta como erro, o sistema não entra em estado erróneo e, portanto, erros não precisam ser detectados, confinados e recuperados. No entanto, em caso de falhas permanentes, a localização e o reparo da falha ainda são necessários [36]. A tabela 3.1 exemplifica os mecanismos normalmente utilizados para implementação de mascaramento de falhas.

Tabela 3.1: Mecanismo para mascarar falhas (Adaptado de [36]).

Mecanismo para mascarar falhas	
Mecanismo	**Aplicado a:**
Replicação de componentes	Qualquer componente de hardware
ECC (código de correção de erros)	Informação transmitida ou armazenada
Diversidade, programação n-versões	Especificação, projetos, programas
Blocos de recuperação	Software

3.5 Gestão de Tolerância a Falhas na Computação na Cloud

Numa plataforma de cloud são identificáveis três camadas: recursos físicos, VMs (virtual machines) e aplicações, conforme é apresentado na figura 3.3. Cada uma delas (camadas) está sujeita a eventuais falhas. Sendo assim, numa plataforma de cloud pode surgir três tipos de falhas: falha de hardware, falha de VMs e falha de aplicação [7].

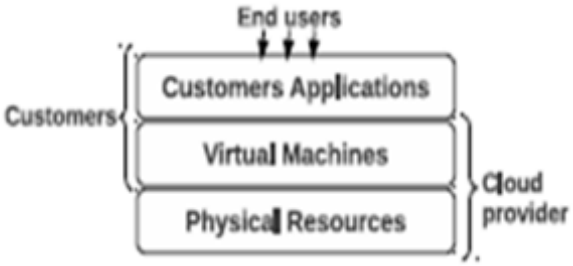

Figura 3.3: Arquitectura de computação na cloud [7].

Uma das dificuldades para implementar os mecanismos de tolerância a falhas numa arquitectura de cloud pode ser resumido por esta pergunta(ver figura 3.4): quais dos intervenientes da cloud (provedor ou cliente) é o mais capaz de gerir as falhas, dependendo de seus direitos de acesso na arquitectura da cloud? Por outras palavras, é razoável deixar exclusivamente a responsabilidade de gerir tolerância a falhas para um dos interveniente de cloud sabendo que: falhas de hardware só podem ser detectadas e reparadas pelo provedor de cloud; falhas de VM podem ser detectadas pelos dois participantes, mas apenas reparadas pelo provedor de cloud; e falhas de aplicação só podem ser detectadas pelo cliente, mas podem ser reparadas pelos dois intervenientes [7].

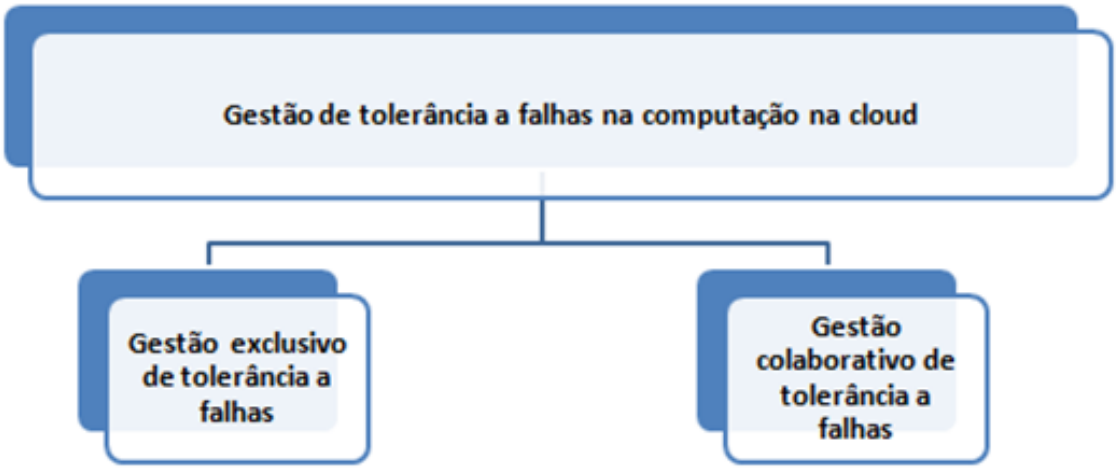

Figura 3.4: Gestão de tolerância a falhas na computação na cloud.

É de realçar que existem duas visões de gestão de tolerância a falhas (de acordo, com esquema apresentado na figura 3.4) O primeiro consiste em dar as responsabilidades de detecção e reparo para um participante da cloud (exclusivamente), enquanto o segundo é aproveitar as habilidades dos dois tipos de participantes (colaborativo). De acordo com essas duas visões, serão

apresentadas algumas técnicas de tolerância a falhas para os três tipos de falhas mencionadas anteriormente: hardware, VM e aplicação [7].

3.5.1 Gestão Exclusiva de Tolerância a Falhas

Nessa secção serão apresentadas as técnicas exclusivas de tolerância a falhas para os três tipos de falhas: aplicação, VM e hardware.

Tolerância a falhas na aplicação

AS falhas de aplicação são detectáveis apenas no nível do cliente. Para cada aplicação, o cliente implanta na cloud componentes de software especiais chamados sensores, que monitoram a aplicação. De acordo com o monitoramento, um sensor pode desencadear a execução de um procedimento para reparar a aplicação, quando esta funcione mal. Os procedimentos para reparar uma aplicação com falha são [7]:

- O primeiro procedimento diz respeito à aplicação "stateless" (como loadbalancers, por exemplo, HAProxy). Este reparo consiste em reiniciar o servidor do sistema defeituoso na mesma VM.

- O segundo procedimento diz respeito a servidores "statefull" (por exemplo, a base de dados MySQL). Neste caso, o cliente deve implementar um mecanismo para salvar o estado do servidor para que possa ser restaurado antes que o servidor seja reiniciado.

Tolerância a falhas nas VMs

As falhas de VM podem ser detectadas e reparadas pelos dois participantes (provedor e cliente) da cloud. Neste nível as políticas de reparo são implementadas exclusivamente por um deles. O reparo da VM com falha é organizado como segue [7]: o nível do cliente solicita que a cloud libere a VM com falha; aloca uma nova VM; implanta e inicia os servidores que estavam em execução na VM com falha; e restaura o estado desses servidores no caso de servidores "statefull" [7].Quando a gestão é feita pelo cliente, cabe a ele implementar o seu próprio sistema de monitoramento, mas é uma tarefa complexa e leva a desperdício de recursos de redes. Esta tarefa está mais indicado para o provedor de cloud, pois tem acesso direto aos hypervisor de VM, visto que é através do hypervisor recolhe informações detalhadas sobre o status da VM, informações que permitem implementar soluções mais precisas de tolerância a falhas [7].

Tolerância a falhas da máquina física: falhas de hardware

As falhas de hardware são detetadas no nível do provedor de cloud. Neste nível, a tolerância a falhas de hardware é implementado com um sistema de monitoramento composto por sensores implantados em diferentes máquinas físicas. Para reparo, o provedor vai iniciar numa nova máquina (ou várias máquinas de acordo com suas capacidades) o mesmo número de VMs, que foram hospedadas na máquina com falha. Além disso, todos os estados da VM devem ser gravados por pontos de salvaguarda (checkpoint) para que a restauração da VM seja possível [7].

3.5.2 Gestão Colaborativa de Tolerância a Falhas

Nessa secção serão relatadas as técnicas colaborativas de tolerância a falhas para também os três tipos de falhas: aplicação, VM e hardware.

Tolerância a falhas na aplicação

As falhas na aplicação são detectadas através da colaboração entre o cliente e o provedor da cloud [7]. As falhas de software (falhas na programação ou aplicação) podem ser exploradas utilizando métodos estáticos e dinâmicos similares àqueles utilizados para o tratamento de falhas de hardware. Um desses métodos é a programação n-version (diversidade), que utiliza redundância estática na forma de programas independentes [5].

Tolerância a falhas nas VMs

No nível do cliente, uma falha da VM detectada por um sensor pode ser devido a uma falha de hardware (máquina que hospeda a VM), que está fora do âmbito do cliente. Uma detecção de falha da VM no nível da cloud permite obter uma decisão mais precisa [7]. Quando a falha é detectada pelo provedor de cloud, o provedor de cloud inicia uma nova VM com os mesmos recursos (rede, memória, CPU, imagem) e em seguida, chama o cliente para reimplantar, reiniciar e sincronizar a nova VM [7].

Tolerância a falhas da máquina física: falhas de hardware

Do ponto de vista do cliente, a falha de uma máquina física é idêntica às falhas da VM. Quando detectada pelo provedor de cloud, tal falha pode ser resolvida de forma colaborativa durante o reparo de cada VM hospedada na máquina com falha [7].

3.6 Elevada Disponibilidade

A alta disponibilidade consiste em manter disponível por meio da tolerância a falhas, isto é, utilizando mecanismos que fazem a detecção, mascaramento e a recuperação de falhas, sendo que esses mecanismos podem ser implementados a nível de software ou de hardware [29]. O objetivo de implementar um sistema de alta disponibilidade, é com o propósito de garantir que o sistema esteja sempre disponível para atender as solicitações do utilizador.

Um outro conceito que também não se pode dissociar de alta disponibilidade é o balanceamento de carga, que é um mecanismo que tem por objetivo atingir a escalabilidade, repartindo a carga de processamento entre duas ou mais máquinas [42]. Uma outra tarefa do balanceamento de carga é promover a melhoria de desempenho do sistema, através da distribuição de tarefas a serem executadas. Isto é, trabalhar de forma que as diferentes máquinas do "Cluster" funcionam como uma única máquina.

A alta disponibilidade pode ser implementada com hardware redundante executando instâncias redundantes de cada serviço. Se uma peça de hardware que executa uma instância de um serviço falhar, o sistema poderá então fazer o "Failover" para usar outra instância de um ser-

viço em execução no hardware que não falhou [38]. Um aspeto crucial da alta disponibilidade é a eliminação de pontos únicos de falha (single points of failure-SPOFs). Um SPOF é uma peça individual de equipamento ou software que causa inatividade do sistema ou perda de dados se falhar. Para eliminar os SPOFs, é analisado se existem mecanismos para redundância de [38]:

- Componentes de rede, como switches e router;
- Aplicações e migração automática de serviço;
- Componentes de armazenamento;
- Serviços de instalações como energia, ar condicionado e proteção contra incêndio.

Os sistemas de alta disponibilidade normalmente atingem uma percentagem de tempo de atividade de 99,99% ou mais, o que equivale a menos de uma hora de tempo de inatividade acumulado por ano. Para conseguir isso, os sistemas de alta disponibilidade devem manter os tempos de recuperação após uma falha de cerca de um a dois minutos [38].

3.6.1 Elevada Disponibilidade na Infraestrutura de Cloud OpenStack

A alta disponibilidade é implementada no OpenStack, com a preocupação de garantir a redundância e a disponibilidade dos serviços existentes. Os serviços do OpenStack são divididos em duas categorias: "Stateless services" e "Stateful services". "Stateless services" é um serviço que fornece uma resposta após o seu pedido e, em seguida não requer mais atenção. Os serviços do OpenStack stateless são: nova-api, nova-conductor, aponte-api, keystone-api, neutron-api e nova-scheduler. Enquanto que "Stateful services" é um serviço em que as solicitações subsequentes ao serviço dependem dos resultados da primeira solicitação, são eles :base de dados e message queue do OpenStack [38].

Os "Stateful services" (serviços com estado) podem ser configurados como ativos/passivos ou ativos/ativos, e são definidos da seguinte forma [38]:

- Configuração ativo/passivo: mantém uma instância redundante que pode ser colocada online quando o serviço ativo falhar, como ilustra a figura 3.5. Por exemplo, o OpenStack grava na base de dados principal enquanto mantém uma base de dados de recuperação que pode ser colocada online se a base de dados principal falhar.
- Configuração ativo/ativo: cada serviço tem um backup, mas faz a gestão do sistema principal e sistema redundante em simultâneo, de acordo com a figura 3.6 . Dessa forma, se houver uma falha, é improvável que o utilizador perceba. O sistema de backup já está online e recebe um aumento de carga enquanto o sistema principal é corrigido, este processo chama-se "Failover".

Após a recuperação do sistema principal, ele é colocado novamente online, isto é, é aplicado um "Failback". Normalmente, uma instalação ativa / ativa para um serviço sem monitoração de estado mantém uma instância redundante e as solicitações são balanceadas por carga usando um endereço IP virtual e um balanceador de carga, como HAProxy.

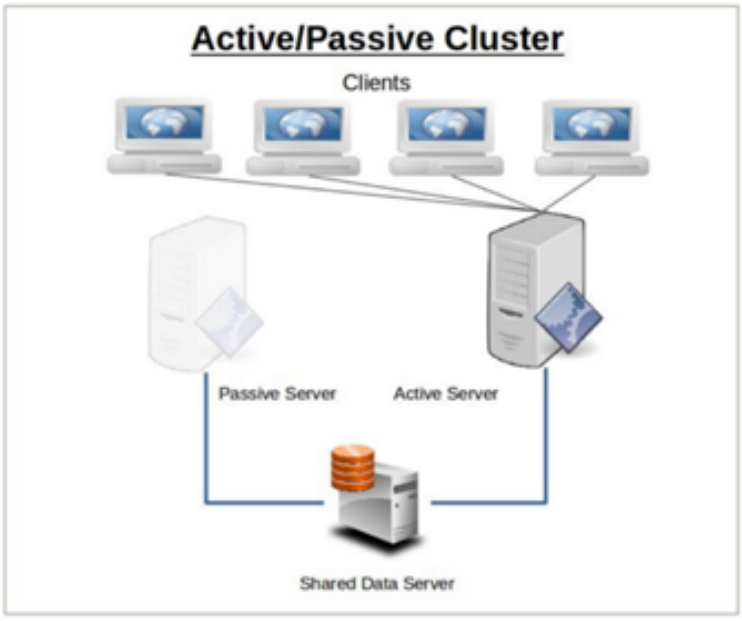

Figura 3.5: Configuração ativo/passivo [37].

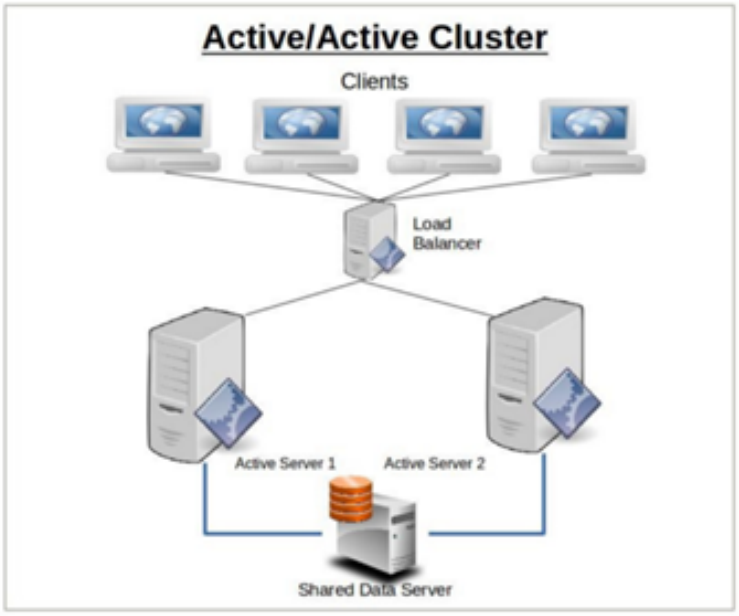

Figura 3.6: Configuração ativo/ativo [37].

3.6.2 Ferramentas de Elevada Disponibilidade

As tecnologias de hardware e software mais comuns que podem ser usadas para criarem um sistema altamente disponível são [38]:

a) Elevada disponibilidade através de redundância de hardware

O uso de tecnologias diferentes para permitir alta disponibilidade no nível de hardware fornece uma boa base para a criação de um sistema de alta disponibilidade. Eis, algumas dessas tecnologias:

- “Redundant switches”: os switches são pontos únicos de falhas, pois a rede é fundamental para operar todos os outros domínios básicos da infraestrutura, como computação e armazenamento.
- “Bonded interfaces”: as “Bonded interfaces” são duas interfaces de rede física independentes tratadas como uma interface no modo de redundância ativa/passiva ou ativa/ativa. No modo ativo/passivo, se ocorrer um erro na interface de rede ativa ou na extremidade remota da interface, as interfaces serão alternadas. No modo ativo/ativo, quando ocorre um erro na interface ou na extremidade remota de uma interface, a interface é marcada como indisponível e deixa de ser usada.
- Balanceadores de carga: os balanceadores de carga física são “routers” especiais que direcionam o tráfego em diferentes direções com base num conjunto de regras. Os balanceadores de carga podem estar no modo redundante de forma semelhante aos comutadores físicos. Os balanceadores de carga também são importantes para distribuir o tráfego para os diferentes componentes ativos/ativos do sistema.
- Armazenamento (storage): a alta disponibilidade do armazenamento físico pode ser obtida com diferentes âmbitos. Alta disponibilidade dentro de uma unidade de hardware com discos redundantes (principalmente organizados em diferentes configurações de RAID); componentes de controlo redundantes, interfaces de E/S redundantes e fontes de alimentação redundantes; e alta disponibilidade em nível de sistema com unidades de hardware redundantes com replicação de dados.

b) Elevada disponibilidade através de ferramentas de software

Ao implementar um sistema de alta disponibilidade, torna-se necessário organizar os servidores num “Cluster”, facilitando assim a utilização de softwares de “cluster management”. Esses softwares são denominados de “Cluster Resource Management (CRM) ”, e permitem detetar falhas nos nós do “Cluster”. As falhas podem ser a nível de hardware ou a nível de serviços. Sempre que ocorra uma falha num dos nós do “Cluster”, ela é detetada pelos softwares de “cluster management” que executam o processo de “Failover” e “Failback”. Para a alta disponibilidade na infraestrutura da cloud OpenStack são recomendados softwares como [38]: HAProxy, Keepalived, Pacemaker etc.

HAProxy

O HAProxy é um serviço do Linux que garante um balanceamento e alta disponibilidade num conjunto de servidores, como também o serviço de ”proxying”, ao não expor diretamente estes na Internet [42]. O HAProxy é um balanceador de carga gratuito para HTTP e TCP. Ele é adequado para o rastreamento da Web sob cargas muito altas. A ferramenta HAProxy através de uma monitorização ativa dos serviços deteta possíveis falhas e direciona automaticamente o tráfego para os restantes nós ativos(ver figura 3.7). Logo pode-se dizer que é uma ferramenta adequada para alcançar a alta disponibilidade e tolerância a falhas. Na instalação do HAProxy deve se acautelar que ele não seja um ponto único de falha, é aconselhável ter várias instâncias do HAProxy em execução. Geralmente, é implementado por padrão nas plataformas de cloud [43]. A forma como o HAproxy funciona facilita a sua integração nas arquiteturas existentes e sem riscos, enquanto oferece a possibilidade de não expor servidores frágeis à rede, como ilustra a figura 3.7.

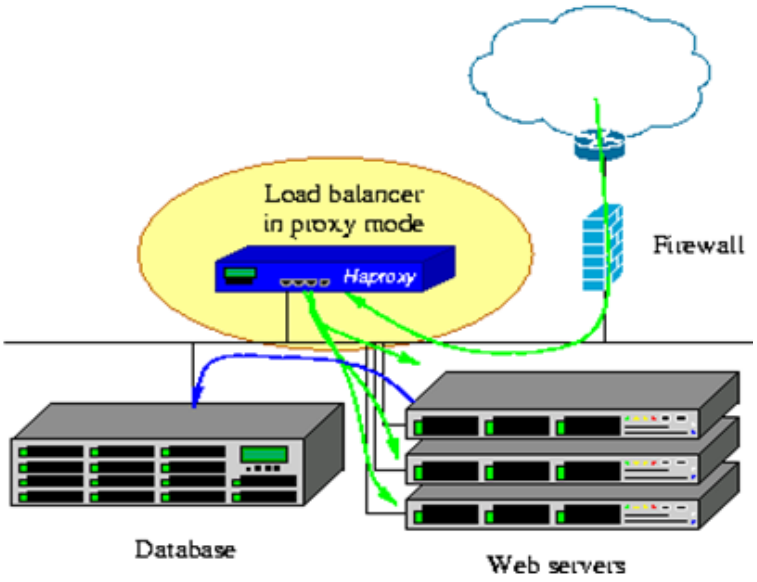

Figura 3.7: Servidor HAProxy [43].

Keepalived

Como foi dito anteriormente o HAProxy faz o balanceamento de carga e assegura a alta disponibilidade dos servidores. Em caso de falha do servidor HAProxy esses serviços já não serão garantidos. Para colmatar essa falha de HAProxy é necessário configurar os servidores HAProxy, isto é, criar um "Cluster" de HAProxy. É neste contexto que entra o software Keepalived. A semelhança do HAProxy é software "Open source". O Keepalived fornece recursos para o balanceamento de carga e a alta disponibilidade para o sistema Linux e infraestruturas baseadas em Linux. A alta disponibilidade com o Keepalived é obtida através do protocolo VRRP (Virtual Router Redundancy Protocol) [44]. O protocolo VRRP executa o processo de "Failover" dinâmico de endereços IP, de um router virtual para o outro, caso haja falhas. Baseado no protocolo VRRP, o Keepalived permite que se implemente um "Cluster " ativo/passivo com servidores HAProxy. Um servidor HAProxy é configurado como "Master" atribuindo-lhe um IP virtual (VIP), que em caso de falha o Keepalived migra automaticamente o VIP para outro servidor HAProxy pertencente ao "Cluster" [45], conforme é ilustrado nas figuras 3.8 e 3.9.

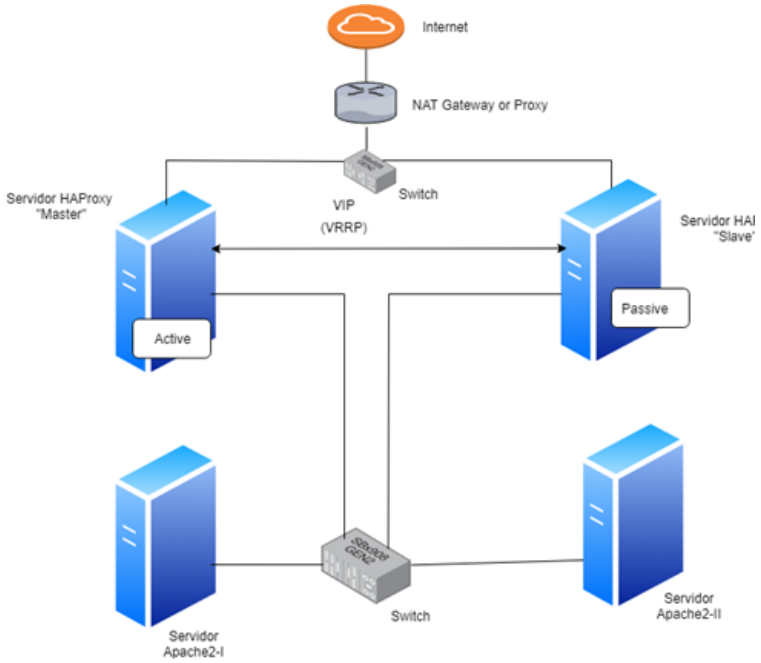

Figura 3.8: Alta disponibilidade de servidores ”HAProxy” com ”Keepalived” : cenário sem falhas (adpatado de [45] [46]).

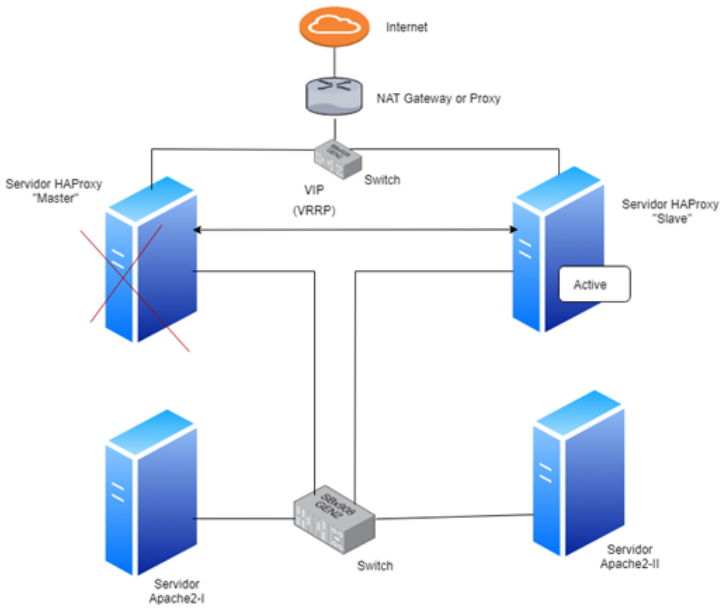

Figura 3.9: Alta disponibilidade de servidores ”HAProxy” com ”Keepalived”: cenário após falha do “HAProxy“ “Master” (adpatado de [45] [46]).

Servidor MySQL

Os servidores MyQL são configurados para que haja uma réplica de dados entre eles. O mecanismo de réplica de dados permite a manutenção de várias cópias idênticas de um mesmo dado nos servidores de base de dados [47] que fazem parte dos “Cluster” da replicação. As

principais vantagens da réplica de dados são a redundância, o que torna sistema tolerante a falhas, também permite um maior balanceamento de carga, já que o acesso pode ser distribuído entre réplicas, e finalmente ter-se-á um backup online dos dados, já que todas as réplicas estariam sincronizadas [47]. Um dos modelos de replicação conhecido é o "Master"-"slave", onde-se tem um servidor atuando como "master" e um ou mais servidores atuando como "slave" (neste trabalho tem se apenas um "slave"). O master grava num log binário todos os comandos de atualizações da base de dados. Desta forma, todas as alterações realizadas no master são imediatamente replicadas para o servidor "slave" [47].

A implementação da réplica de servidores traz um ganho notável que é a alta disponibilidade, ou seja, se um servidor falhar tem outro servidor para retomar o serviço.Para finalizar é demonstrado o funcionamento da réplica dos servidores ("Master-slave"). O MySQL realiza a replicação num processo de três fases [47], conforme é ilustrado na figura 3.10, de uma forma detalhada:

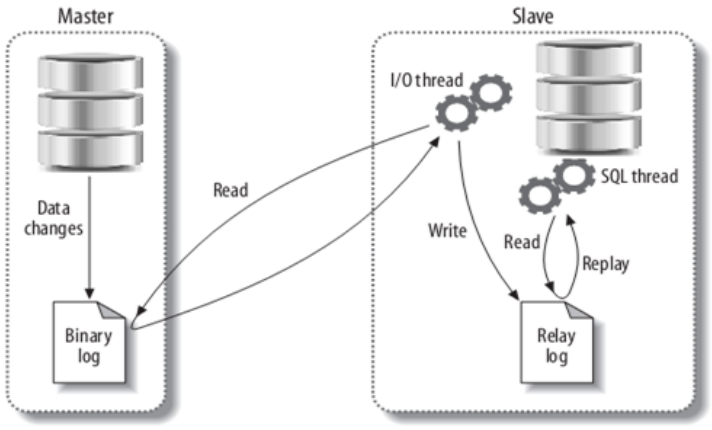

Figura 3.10: Replicação dos servidores do MySQL (adaptado de [47]).

Pacemaker

O pacemaker é um software "open-source" da ClusterLabs [48]. O pacemaker pode ser definido como um software de recuperação de falhas a nível de serviço. Normalmente ele é utilizado em parceria com outros softwares que fazem os registros dos nós e troca de mensagens entre os outros nós do "cluster" [49]. Os Softwares que podem ser integrados com pacemarker são [29]:

- Corasync: surgiu do projeto OpenAIS e é responsável pelo processo de registro dos nós e pelo processo de "Failover" [50].

- Hearbeat: é um software "open-source" responsável pelo envio de mensagens entre os nós do cluster, para além de inicializar e finalizar os serviços [51].

Observando a figura 3.11, percebe se que na camada inferior estão localizados os nós do "Cluster". Na segunda camada estão os softwares de envio de mensagens e na terceira camada encontra se o pacemarker. Nas últimas camadas estão os serviços que serão executados no "Cluster".
De uma resumida as funcionalidades do pacemaker são [29]:

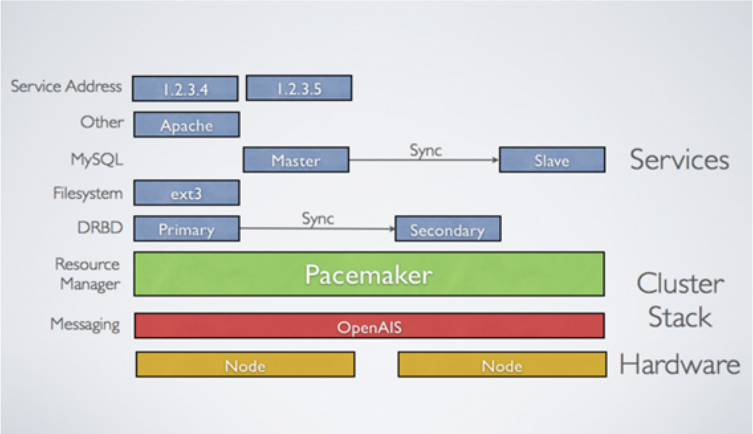

Figura 3.11: Pacemaker [48].

- Iniciar e finalizar os serviços dos nós do "Cluster". Os serviços podem ser um servidor web, interface de rede, ou uma máquina virtual;
- Replicação de configuração do "Cluster", a configuração alterada num nó pode ser replicada para todos os demais nós de uma forma transparente;
- Eleição de um nó como primário, isto é, no caso de uma falha no nó primário, um outro nó será eleito primário de forma transparente.

3.7 Conclusão

Tendo em conta que qualquer sistema computacional, está sujeito a falhas, este capítulo apresentou alguns conceitos sobre a tolerância a falhas, bem como as técnicas para as gerir. Em particular explanou alguns conceitos sobre alta disponibilidade, destacando as tecnologias para a implementar, tanto a nível de hardware como software.

Capítulo 4

Ambiente Experimental e Ferramentas de Avaliação de Desempenho

Este capítulo debruça-se sobre a instalação da infraestrutura de cloud OpenStack, apresenta uma aplicação web como caso de estudo e descreve a construção de elevada disponibilidade para essa aplicação. Ainda no decorrer do capítulo, são apresentadas as ferramentas para a análise do desempenho da aplicação, tanto do lado cliente como do lado do servidor.

4.1 Instalação e Configuração da Cloud OpenStack

Nesta secção serão apresentados os detalhes para a implementação da infraestrutura de cloud privada Openstack.

4.1.1 Ambiente de Instalação

O ambiente é composto por três máquinas (máquina 1, máquina 2 e máquina 3), com o sistema operativo Linux Ubuntu 14.04 LTS e um switch de 8 portas. Na máquina 1 foi instalado o nó "Controller", na máquina 2 o nó "Compute" e na máquina 3 está alojado o nó "Block Storage". Todas as máquinas (excepto a máquina 3 que tem apenas uma placa de rede) foram configuradas com duas placas de rede, uma placa para rede privada (Management network) e outra para a rede do provedor (Provider network). As características das máquinas e os dados da configuração estão nas tabelas 4.1, 4.2 e 4.3 respetivamente. Terminada a instalação do sistema operativo Linux, foi instalado nos três computadores o OpenStack, versão "Mitaka".

Tabela 4.1: Características e dados da configuração da máquina 1 ("Controller").

Controller	
Memória	16 GB
Processador	Intel(R) core(TM) i7 CPU860280GHZ x8
Disco	500 GB
Management network (eth1)	10.0.6.209
Provider network (eth0)	Não numerado

Tabela 4.2: Características e dados da configuração da máquina 2 ("Compute").

Compute	
Memória	8 GB
Processador	Intel(R) core(M) i7 CPU860280GHZ x8
Disco	500 GB
Management network (eth0)	10.0.6.208
Provider network (eth1)	Não numerado

Tabela 4.3: Caraterísticas e dados da configuração da máquina 3 (“Block Storage”).

Block Storage	
Memória	4 GB
Processador	Intel(R) core(M) i7 CPU860280GHZ x8
Disco	500 GB
Management network (eth0)	10.0.6.213

4.1.2 Arquitetura e Configuração de Rede

A Figura 4.1 exibe a topologia da rede, onde o nó intitulado de “Controller” é o gestor da cloud, isto é, executa os serviços de gestão (keystone, horizon, etc.) necessários para o funcionamento do OpenStack.

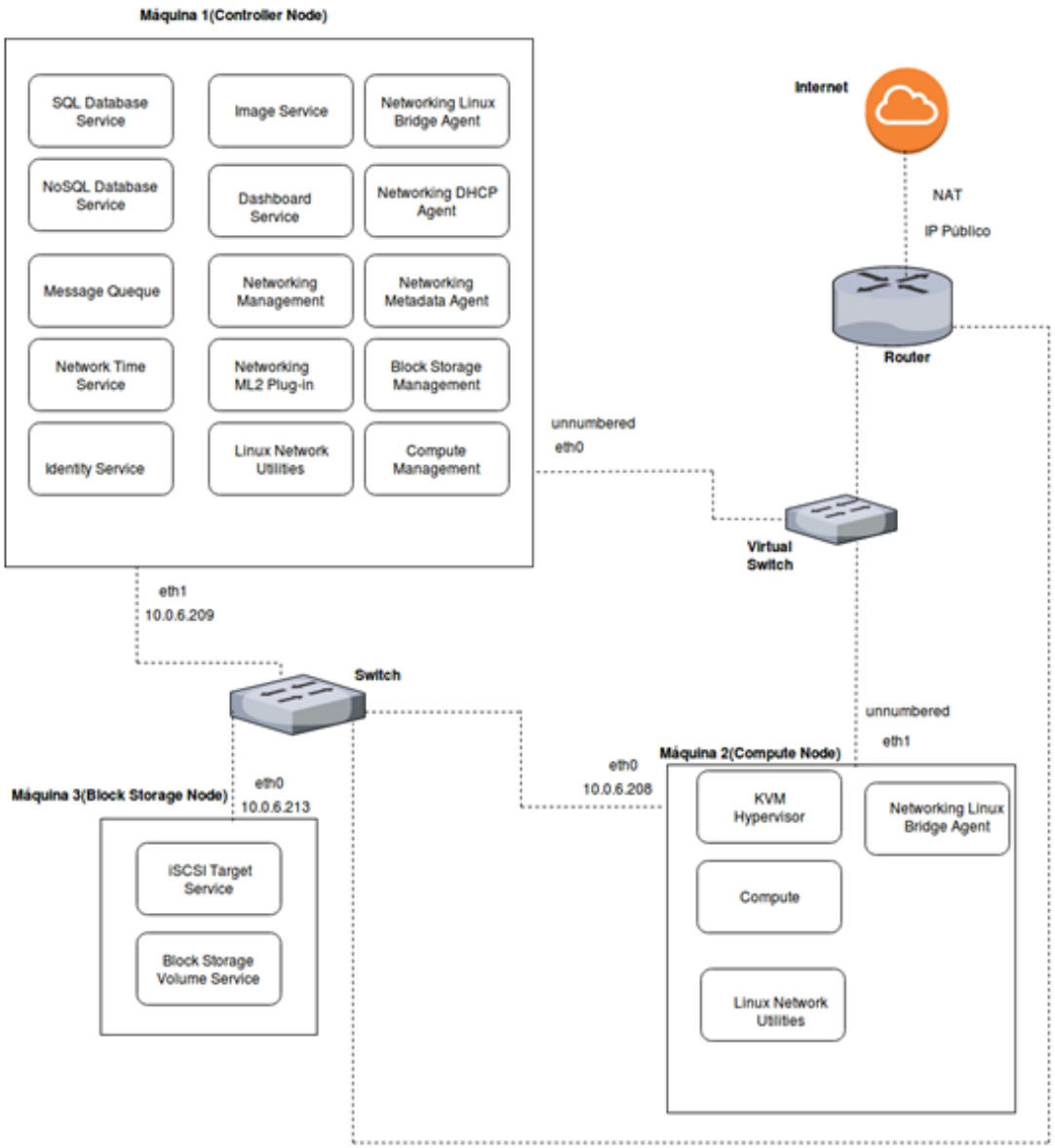

Figura 4.1: Arquitetura e configuração de rede.

O nó “Compute” executa as instâncias de máquina virtual no OpenStack e o nó “Block Storage”

(serviço de cinder) fornece dispositivos de armazenamento em bloco para as instâncias. A maioria dos serviços foram instalados no nó "Controller", portanto, se ele falhar, todo o ambiente fica comprometido, deixando assim a cloud fora de operação até que seja detetado e resolvido o problema. Para resolver esta situação, deve-se implementar um mecanismo de alta disponibilidade, por exemplo o software pacemaker, que faça a gestão da réplicação dos nós. Isto é, se um nó falhar o outro nó do "Cluster" retoma os serviços.

4.1.3 Instalação e Configuração dos Nós da Infraestrutura da Cloud OpenStack

Os principais passos para instalar e configurar foram:

- O "OpenStack packages" foi instalado em todos os nós;
- Todos os nós devem estar sincronizados no mesmo fuso horário, caso contrário o serviço Cinder-volume do nó "Block Storage" não funciona;
- As máquinas que contêm os nós da infraestrutura da cloud OpenStack devem estar configuradas na mesma rede;
- A versão do Ubuntu que é compatível com OpenStack "Mitaka" é Ubuntu 14.04 LTS.

Para instalar e configurar os nós da infraestrutura da cloud OpenStack e os seus respetivos serviços, basta seguir as etapas da instalação do site oficial do OpenStack[1], neste caso a versão "Mitaka". Todavia, a instalação é muito complexa porque alguns serviços, como keystone e neutron do nó "Controller", geram erros de configuração e a forma de corrigi-los não está no manual de instalação do OpenStack. A forma de ultrapassar esses erros de configuração e alguns detalhes essenciais da instalação e configuração dos nós da infraestrutura da cloud OpenStack serão descritos a seguir:

Instalação e configuração do nó "Controller": este é o primeiro nó a ser instalado e configurado, pois é neste nó que se encontra o serviço "keystone" que é crucial na infraestrutura cloud OpenStack, visto que é responsável pela autenticação dos outros serviços e autorização para aceder aos mesmos. Se o utilizador optar por instalar outros serviços como, por exemplo, "glance" ou "neutron" antes de instalar "keystone", vai originar erro de autenticação, pois estes ou outros serviços devem ser reconhecidos pelo keystone e só em seguida prosseguir com as suas instalações. Durante a instalação do "keystone", geralmente, gera-se o erro na inserção de dados na sua respectiva base de dados. Para verificar se os dados da configuração foram inseridos com sucesso na base dados do keystone, executa-se o servidor MySQL e aplica-se o comando: **>Show database keystone**; se a base de dados estiver vazia, repete-se o comando para inserção de dados **# su -s /bin /sh -c "keystone-manage db_sync" keystone**. Se o erro persistir, a alternativa é remover a base de dados e criá-la novamente. O outro serviço do "Controller" que também requer atenção na sua instalação é o serviço de "neutron".O seu erro é igual ao do "keystone", isto na inserção de dados na sua base de dados, mas a forma de resolver é diferente. Para ultrapassar este erro de configuração deve-se comentar a seguinte linha de código: **connection=sqlite:////var/lib/neutron/neutron.sqlite** (geralmente está localizada na linha 707) da secção **[Database]** do ficheiro de configuração: **/etc/neutron/neutron.conf**. Os serviços de base de dados: SQL Database e NoSQL Database são fáceis de configurar, embora as suas "passwords" sejam solicitadas várias vezes durante o processo de instalação, principalmente a "password" do SQL Database. Os outros serviços, nomeadamente "glance" (image

[1] https://docs.openstack.org/mitaka/install-guide-ubuntu/

service), “compute management”, “dashboard” e “block store management” não acarretam problema no processo de instalação.

Instalação e configuração do nó “Compute” :o nó “Compute” é (ou pode ser) configurado em simultâneo com o nó “Controller” após a instalação do keystone. É aconselhável instalar o software de virtualização, isto é, o hypervisor em primeiro lugar, neste caso o KVM (Kernel-based Virtual Machine) que é recomendado pelo OpenStack versão “Mitaka”. O KVM é um software open-source de virtualização para Linux em arquitecturas x86. Para verificar se uma máquina suporta a virtualização com KVM é aplicado o seguinte comando: **# egrep -c ’(vmx|svm)’ /proc/cpuinfo** . Se o output for 0 (zero) é porque a máquina não suporta a virtualização com KVM, se for um número superior a zero, logo a máquina suporta a virtualização com KVM e não precisa de configurações adicionais. A seguir são instalados os serviços da “nova” e os serviços de “neutron”.

Instalação e configuração do nó “Block Storage”: o nó “Block Storage” também pode ser instalado em simultâneo com outros nós (após a instalação do keystone no nó “Controller”). Neste nó é instalado o serviço cinder-volume, serviço esse que fornece o armazenamento às máquinas virtuais. Mas antes disso é preciso formatar o disco para criar volumes no formato LVM (Logical Volume Manager). O comando aplicado para formatar e escolher o formato do disco é: **# pvcreate /dev/sdb** . Se o disco da máquina não reconhecer este comando, a alternativa é fazer a partição com o software GParted.

Configuração das redes para as máquinas virtuais: o processo da configuração das redes para as máquinas virtuais requer atenção, pois no manual da configuração do OpenStack não está muito explícito. Devem-se configurar duas redes: uma rede do provedor em que o seu endereço será igual à rede física sobre a qual estão configurados os nós da Infraestrutura da cloud OpenStack e uma rede interna para conectar as máquinas virtuais. A rede interna estabelece a comunicação entre as máquinas virtuais. A interação entre a rede interna e a rede do provedor é feita através de um router virtual. É graças a essa interação que as máquinas virtuais têm acesso à Internet.

4.2 Aplicação Exemplo

A aplicação na cloud é uma página web desenvolvida em HTML, PHP e MySQl que permite a inserção e consulta de dados numa base de dados. Ela (aplicação) está hospedada num servidor web (Apache 2), que por sua vez, está alojado numa máquina virtual, denominada de máquina virtual 1 (VM 1), máquina virtual esta, que está alojada na infraestrutura da cloud descrita na secção 4.1. Isto é, foi instanciada na máquina 2 denominada de nó “Compute”. Também na VM 1 foi instalado o servidor MySQL ao qual a página web (mencionada anteriormente) acede. A página web é acedida através dos seguintes URLs: http://192.168.1.4/aluno.ubi/ ou http://10.0.6.4/aluno.ubi/, pois a VM 1 (e os servidores nelas alojados) tem dois endereços IPs, um interno e outro flutuante[2] . Ou seja, ela está conectada a duas redes, uma interna e outra externa. Para se comunicarem com a rede externa vão utilizar o IP flutuante. Esta descrição pode ser melhor percebida, através da figura 4.2.

[2]Um IP flutuante é um endereço IPv4 público que pode ser associado dinamicamente pelo utilizador a uma instância (Máquina virtual gerida pelo OpenStack).

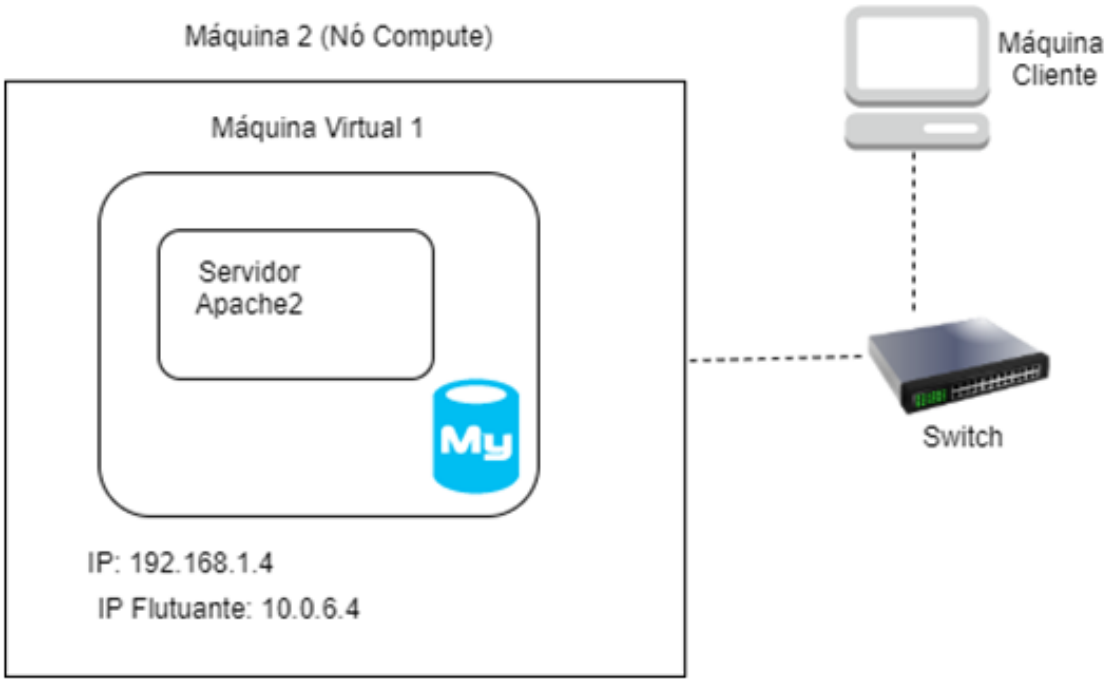

Figura 4.2: Cenário da aplicação na cloud.

A característica da máquina virtual 1 e seus os dados da configuração estão na tabelas 4.4:

Tabela 4.4: Características e dados da configuração máquina virtual 1 (VM 1).

Máquina virtual 1	
Memória	2 GB
Processador	Intel(R) core(M) i7 9xx(Nehalem, core I7, IBRS update)-(1vCPU)
Disco	103,4GB GB
IP	192.168.1.4
I IP Flutuante	10.0.6.4
Sistema Operativo	Ubuntu 14.04 LTS

4.3 Aplicação Exemplo num Cenário de Elevada Disponibilidade

O cenário da alta disponibilidade foi implementado na máquina 2(nó "compute") da seguinte forma: foi instanciada uma outra máquina virtual (máquina virtual 2-VM 2) na mesma máquina física, conforme ilustra a figura 4.3, ficando assim o sistema da cloud com duas máquinas virtuais. Nesta nova máquina virtual (VM 2), está alojada uma cópia da aplicação exemplo descrita na secção anterior. Esta réplica pode ser acedida com os endereços URL: http://192.168.1.5/aluno.ubi/ ou http://10.0.6.5/aluno.ubi/, ou seja, tem dois endereços IPs, um interno e outro flutuante, pois, ela está conectada a duas redes, uma interna e outra externa. O IP flutuante permite que ela seja acedida externamente. A aplicação tem as mesmas funcionalidades que a anterior (aplicação alojada na VM 1). Os servidores MySQL, foram configurados para que haja uma réplica de dados entre eles, isto é, todas as operações realizadas no master são imediatamente replicadas para o servidor "Slave". Na máquina virtual 1 (VM 1) está o servidor MySQL "Master" e na máquina virtual 2 (VM 2) está o servidor MySQL "Slave". O servidor HAProxy está alojado

na máquina física 2 (nó compute), como ilustra a figura 4.3. É de realçar que o cliente não acede a essas páginas através das URLs acima mencionadas, mas através da URL associada ao servidor HAProxy (http://10.0.6.208/aluno.ubi/). Isto é, todas as solicitações provenientes da Internet (http request) por parte dos utilizadores para acederem às páginas são atendidas pelo Servidor HAProxy, que, por sua vez, reencaminha o tráfego, ou seja, o processo de requisição para um dos servidores que contém a aplicação solicitada de acordo com as suas disponibilidades, isto é, o HAProxy faz o balanceamento de carga entre os servidores. Em seguida, o mesmo servidor HAProxy encarregar-se-á de devolver o pedido (http response) feito pelos utilizadores através da URL que está associada ao IP da máquina onde está instalado, isto é, http://10.0.6.208/aluno.ubi/. A conexão entre as máquinas virtuais que contêm os servidores apache2 e a máquina física que contém o servidor HAProxy é feita através de um switch virtual.

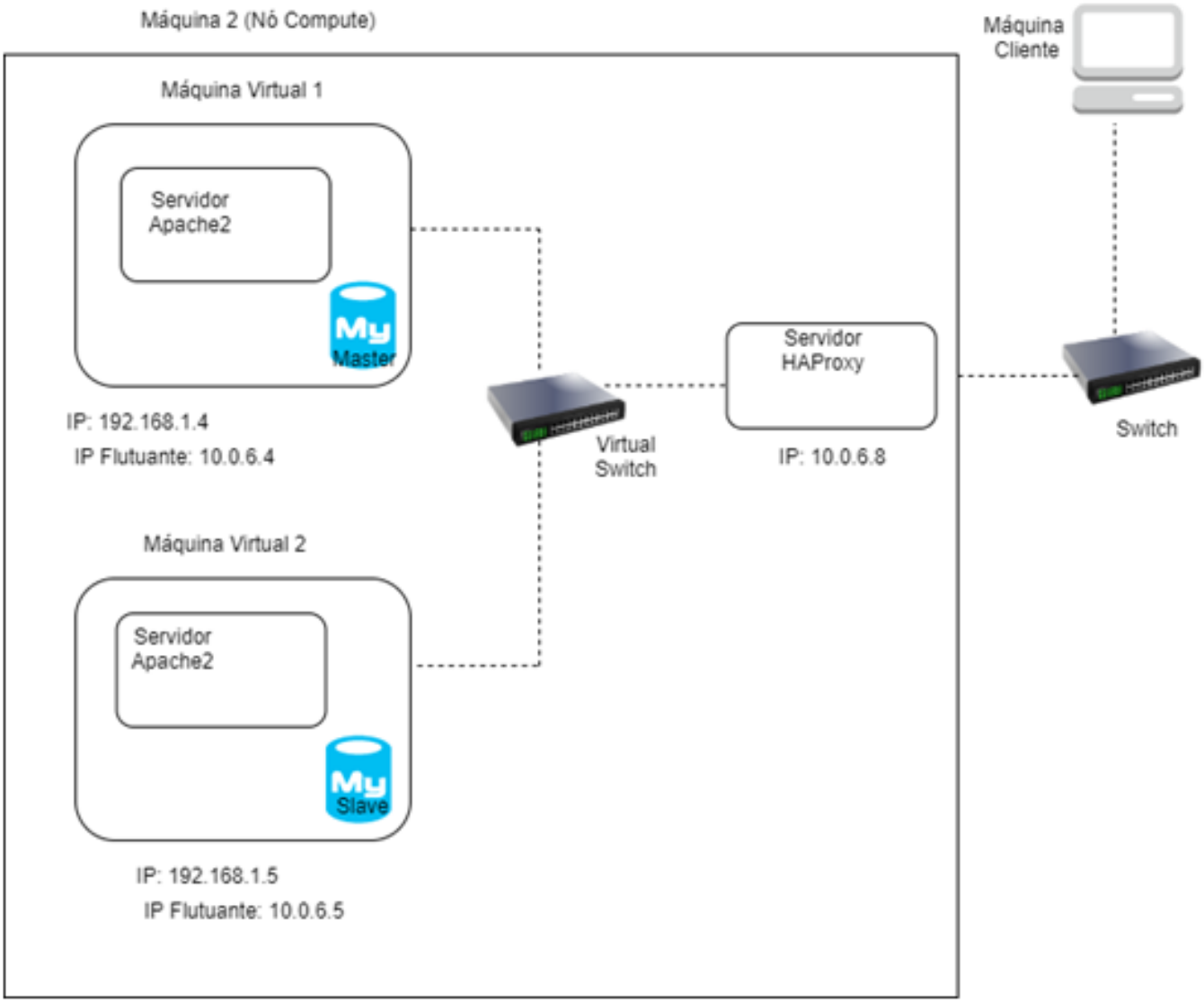

Figura 4.3: Cenário de Alta Disponibilidade.

As características da máquina virtual 2 (VM 2) deste cenário da alta disponibilidade e os dados

da configuração estão na tabela 4.5:

Tabela 4.5: Características e dados da configuração máquina virtual 2 (VM 2).

Máquina virtual 2	
Memória	2 GB
Processador	Intel(R) core(M) i7 9xx(Nehalem, core I7, IBRS update)-(1vCPU)
Disco	103,4GB GB
IP	192.168.1.5
I IP Flutuante	10.0.6.5
Sistema Operativo	Ubuntu 14.04LTS

Esta configuração garante a disponibilidade dos servidores web. O funcionamento do sistema deste trabalho foi arquitetado com o propósito de que se um servidor falhar o outro retome o serviço imediatamente, tudo isso graças à distribuição que é feita pelo HAProxy. Se o HAProxy não consegue conectar-se num nó, direciona automaticamente o tráfego para outro nó ativo. Ou seja, sempre que haja falha de um dos servidores alojados nas máquinas virtuais é aplicado o processo de "Failover". Mas se a falha for no servidor alojado na máquina física, neste caso, no HAProxy, todo o sistema deixará de funcionar, pois o sistema não foi configurado para eventuais falhas deste servidor, isto é, não tem outro servidor para retomar os seus serviços.

4.4 Ferramentas de Avaliação de Desempenho

As ferramentas existentes para medir o desempenho de um website e servidores podem ser agrupadas em dois tipos: "client side" (lado do cliente) e "server side" (lado do servidor), cada uma delas com o seu objetivo diferente, isto é, "client side" testa o desempenho alusivo às métricas (tempo de carregamento e tempo de conexão) de acesso à página por parte das máquinas dos utilizadores (clientes) e a ação do "server side" recai sobre o comportamento do servidor onde o website se encontra alojado, baseia-se nas métricas de tempo de resposta e tempo de conexão.

4.4.1 Lado do Cliente

As ferramentas do lado do cliente, como já referido acima analisam o comportamento em relação às métricas (tempo de carregamento e tempo de conexão) de acesso à página por parte dos utilizadores. Isto é, têm o objetivo primordial de quantificar as métricas que englobam o lado do cliente, nomeadamente a anáilse do tempo de carregamento permite estudar em detalhe os vários componentes da página . Existem vários sites que disponibilizam essas ferramentas Client side. Muitas delas são gratuitas, tais como: PageSpeed Insights, WebPageTest, Pingdom Website Speed Test, GTmetrix, Mobile Friendly. Neste estudo, a ferramenta escolhida, conforme ilustra a figura 4.4, foi a WebPageTest, pois é uma ferramenta Open Source e com um output fácil de se analisar. Foi inicialmente desenvolvida pela AOL num projeto interno e em 2008 foi aberto sob a licença BSD [3].

O WebPageTest recorre a três parâmetros para realizar o teste: o caminho para o website em estudo, a localização da máquina onde serão realizados os testes e, por último, o tipo de brow-

[3]https://blog.mxcursos.com/5-ferramentas-gratuitas-para-avaliar-o-desempenho-do-seu-site/.

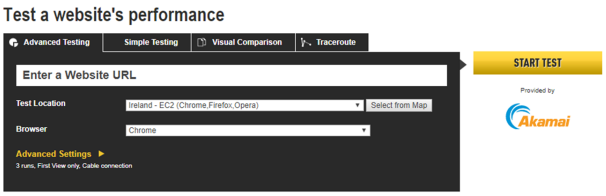

Figura 4.4: Interface do Webpagetest [52].

ser a ser usado como forma de acesso ao website em estudo. Esta ferramenta, por padrão, realiza três vezes o mesmo teste de forma a remover erros presentes nos resultados gerados.

As variáveis medidas pela ferramenta webpagetest são: tempo de carregamento (load time) que indica o tempo total necessário para que uma página web se torne totalmente navegável e o tempo de conexão (First Byte) que retrata o tempo necessário à conexão entre o cliente e o servidor HTTP. O tempo de conexão está muito assente no First Byte que define o tempo necessário o que o servidor demora para entregar o primeiro byte do seu site ao navegador web. Esta métrica também é conhecida por Time To First Byte (TTFB). O tempo de carregamento é medido em segundos e o tempo do First Byte é medido em milissegundos [53].

4.4.2 Lado do Servidor

Conforme foi dito na secção anterior, as ferramentas do lado do servidor ("server side") têm como objetivo estudar o comportamento do servidor onde o website se encontra alojado. O foco destas ferramentas baseia-se no número de pedidos, sendo o item mais importante a análise do tempo de resposta, ou seja, é feita a análise ao comportamento do tempo de resposta consoante o número de pedidos feitos ao website. Este tipo de teste depende muito da capacidade computacional da máquina onde se encontra alojado o servidor do website. Na internet encontram-se disponibilizadas várias ferramentas Server side, tais como: Locust.io [54], Bees with Machine Guns [55], Siege [56], Apache Bench [57], HttpPerf [58],JMeter [59].

A escolha para este estudo recaiu sobre Apache Bench dada a sua vertente open source e a sua enorme popularidade neste tipo de análises. O Apache Bench é uma ferramenta de teste "Server side" originalmente criada para testar o Apache HTTP Server, mas hoje em dia permite a análise de qualquer webserver. É uma ferramenta sem interface gráfica cuja principal métrica de estudo é o tempo de resposta de um website, o que implica a relação entre o número de pedidos feitos ao servidor em simultâneo e o seu tempo de resposta. Também é indicada para o caso em que não é possível ter acesso direto ao servidor em análise. O manuseio desta ferramenta funciona da seguinte forma: O Apache Bench recorre a uma linha de comandos como forma de input, conforme é apresentado na figura 4.5, em que ab representa a chamada da ferramenta Apache Bench, o parâmetro -n representa o número de utilizadores simulados pela ferramenta, o parâmetro -c representa o número de pedidos feito por cada utilizador e, por último, o parâmetro "htttp://10.0.6.208/aluno.ubi/" representa o website em estudo.

```
ab -n 100 -c 10 "http://10.0.6.208/aluno.ubi/
```

Figura 4.5: Exemplo do comando no Apache Bench.

Dependendo do número de pedidos, a análise pode ter a duração de alguns segundos até a algumas horas, sendo que em qualquer dos casos o output é sempre semelhante ao da figura 4.6. Como, neste trabalho, o foco é avaliar o comportamento do sistema perante uma situação de falhas, os valores de maior importância são o "Time per Request" dos vários clientes que permitem o cálculo da métrica tempo de resposta, sendo também importante os valores de "Connection Times" que permitem o cálculo de outra métrica, o tempo de conexão. Esta outra métrica refere-se ao tempo necessário para conexão entre o cliente e o servidor sem haver transferência de dados.

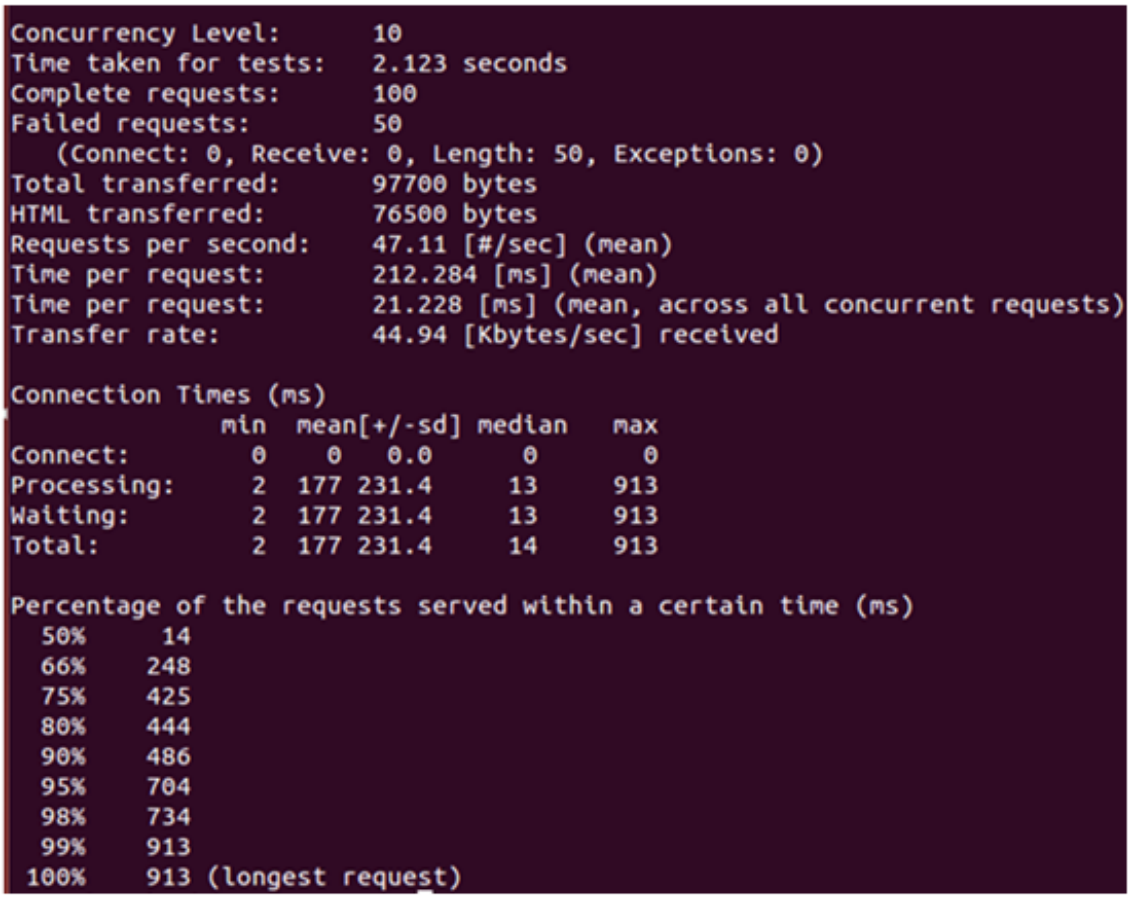

Figura 4.6: Exemplo do Output do Apache Bench.

O Apache Bench foi instalado e executado na máquina física 3. Neste trabalho, em relação a "server side" serão analisados os resultados de duas métricas, tempo de resposta ("Time per request") e tempo de conexão ("Connection Times"). Os valores de ambas as métricas dependem do número de pedidos feitos ao HTTP server.

Capítulo 5

Resultados

Neste capítulo é feita a avaliação do desempenho da aplicação web com as ferramentas mencionadas anteriormente. Após o término da implementação e configuração de aplicação web foram realizados os testes com as ferramentas de avaliação de desempenho do website. Os testes foram efetuados com o intuito de analisar o comportamento da aplicação, em relação aos tempos de acesso à página por parte dos utilizadores e da resposta dos servidores. Os testes para a avaliação do desempenho foram realizadas em duas fases: na primeira fase, foi cronometrado (com as ferramentas de avaliação) o desempenho da aplicação sem tolerância falhas e na segunda fase foi avaliado o desempenho com tolerância a falhas (réplica dos servidores). No caso das métricas do lado do servidor, tanto na primeira fase como na segunda fase, foram medidos os tempos de resposta e conexão, para tal, estabeleceram-se dois intervalos, um intervalo de 500 pedidos a 11000 pedidos, com escala de 500 e outro intervalo de 15000 pedidos a 35000 pedidos, desta feita, com escala 5000. Os dados da medição estão sintetizados numa tabela com valores entre 500 a 10500 pedidos, com intervalos de 1000. Apresentam-se sob a forma de grafica os tempos de resposta em função de número de pedidos para o intervalo de 500 a 10500 e para o intervalo entre 15000 e 35000.

5.1 Avaliação do Desempenho sem Tolerância a Falhas

Nesta secção é avaliado o desempenho da aplicação na cloud, sem qualquer redundância. Esta avaliação é feita em duas situações: no primeiro caso, a aplicação é avaliada tal como ela foi implementada inicialmente, isto é, é avaliada a "aplicação exemplo". Em seguida é avaliado o cenário "aplicação exemplo" com acréscimo do servidor HAProxy. O servidor HAProxy permitirá mais tarde executar o balanceamento de carga nos pedidos à aplicação.

5.1.1 Avaliação da "Aplicação Exemplo"

Desta feita, começa-se por avaliar o cenário sem tolerância a falhas, onde, não está disponível um servidor web nem um servidor de base de dados MySQL para retomar os serviços em caso de falha de um desses servidores, portanto se algum componente falhar a aplicação deixará de funcionar (ver figura 4.2). Neste caso , a VM 1 é um ponto único de falha .

Em relação às métricas do lado do cliente, o tempo de carregamento (load time) e tempo de conexão (first byte) foram de 0,194 segundos (s) e 0,174 milissegundos (ms) respetivamente, conforme pode ser observado na tabela 5.1.

Tabela 5.1: Avaliação do desempenho (lado do cliente) sem tolerância a falhas.

Avaliação do desempenho (lado do cliente) sem tolerância a falhas	
Tempo de carregamento (Load Time)	Tempo de conexão (First Byte)
0,188 segundos	0,168 milissegundos

No que tange às métricas do lado do servidor foram cronometradas as métricas de tempo de resposta e tempo de conexão. A figura 5.1 demonstra os resultados obtidos da relação entre o número de pedidos feito ao servidor e o tempo de resposta e de conexão dos mesmos, de acordo com intervalo descrito .

Número de pedidos	500	1500	2500	3500	4500	5500	6500	7500	8500	9500	10500
Tempo de resposta (ms)	2,61	3,44	5,35	6,63	9,79	12,14	13,49	15,65	18,50	21,65	24,65
Tempo de conexão (ms)	3,00	3,00	5,00	7,00	10,00	12,00	13,00	15,00	18,00	21,00	24,00

Figura 5.1: Avaliação do desempenho (lado do servidor) sem tolerância a falhas.

De acordo, com a figura 5.1, pode-se verificar, que à medida que o número de pedidos aumenta, o tempo de resposta aumenta.A média aritmética do tempo de respostas para o conjunto de observações entre 500 a 11000 pedidos com intervalos de 500 é de 12,9 milissegundos Esta situação pode ser melhor entendida através dos gráficos da figura 5.2, que ilustra a relação número de pedidos e tempo de resposta para os dois intervalos considerados.

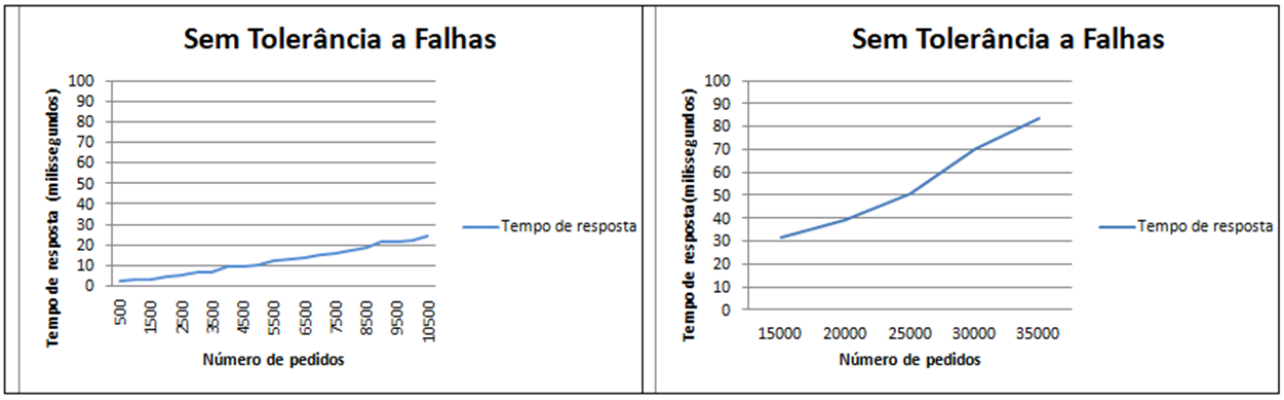

Figura 5.2: Resposta do servidor sem tolerância a falhas.

5.1.2 Avaliação do "Aplicação Exemplo" com HAProxy

Este cenário é quase idêntico ao cenário da situação sem tolerância a falhas (subsecção 5.1.1), também é composto por um servidor web e um servidor MySQl, ambos alojados na VM 1, mas foi acrescentado um servidor HAProxy, servidor esse que está alojado na máquina 2 (nó "com-

pute"), de acordo com a figura 5.3. Percebe-se que este cenário, difere da situação anterior pelo simples facto da existir a conexão entre os servidores alojados na VM 1 com o servidor HA-Proxy. Logo, pode-se dizer, que o propósito da medição dos tempos é de analisar a influência do servidor HAProxy.

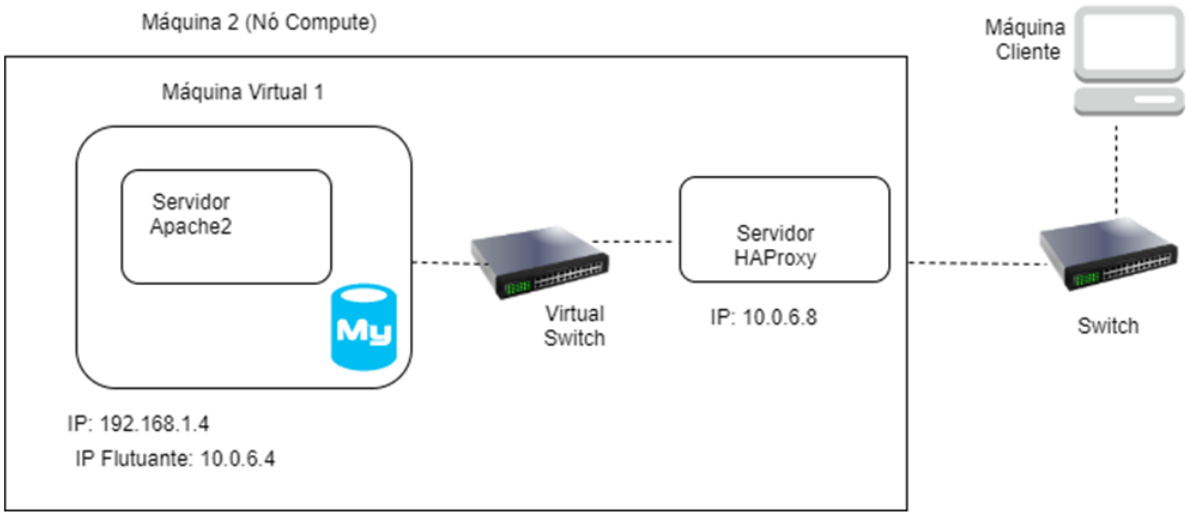

Figura 5.3: Cenário sem tolerância a falhas com HAProxy.

Os valores registados referentes às métricas do lado de cliente são: 0,199 segundos (s) para o tempo de carregamento (Load Time) e 0,168 milissegundos (ms) para o tempo de conexão (First Byte),conforme ilustra a tabela 5.2. Em relação ao cenário anterior, isto é, sem o servidor HAProxy o tempo de carregamento aumentou 0,011 segundos, mas o tempo de conexão manteve-se igual.

Tabela 5.2: Avaliação do desempenho (lado do cliente) sem tolerância a falhas com HAProxy.

Avaliação do desempenho (lado do cliente) sem tolerância a falhas com HAProxy	
Tempo de carregamento (Load Time)	Tempo de conexão (First Byte)
0,199 segundos	0,168 milissegundos

Os resultados do teste das métricas do lado do servidor obtidos nesta situação (sem tolerância a falhas com HAProxy) é demonstrada na figura 5.4 através de uma tabela. Neste cenário a média aritmética do tempo de resposta subiu aproximadamente 5,0%, comparando com o cenário anterior, isto é, desta feita é 13,55 milissegundos.

Número de pedidos	500	1500	2500	3500	4500	5500	6500	7500	8500	9500	10500
Tempo de resposta (milissegundos)	3,53	4,59	6,32	8,21	9,55	11,58	15,58	16,08	18,82	22,41	25,46
Tempo de conexão (milissegundos)	4,00	5,00	6,00	8,00	9,00	12,00	15,00	16,00	19,00	22,00	25,00

Figura 5.4: Avaliação do desempenho (lado do servidor) sem tolerância a falhas com HAProxy.

Comparando os resultados obtidos, do lado do servidor, nesta hipótese com os resultados da

"Aplicação exemplo sem tolerância a falhas e sem HAProxy", verifica-se, que neste cenário, o tempo de resposta é sempre superior. Logo, pode-se concluir que não é viável, ter um cenário com HAProxy sem a réplica dos servidores, seria um investimento sem nenhuma vantagem, pois, a VM 1 continua a ser ponto único de falha e por conseguinte o HAProxy contribui para um ligeiro atraso das respostas às solicitações. Os dados da figura 5.4, podem ser visualizados de uma forma gráfica através na figura 5.5. do lado esquerdo. Do lado direito da figura temos o gráfico para os pedidos entre 15000 e 35000.

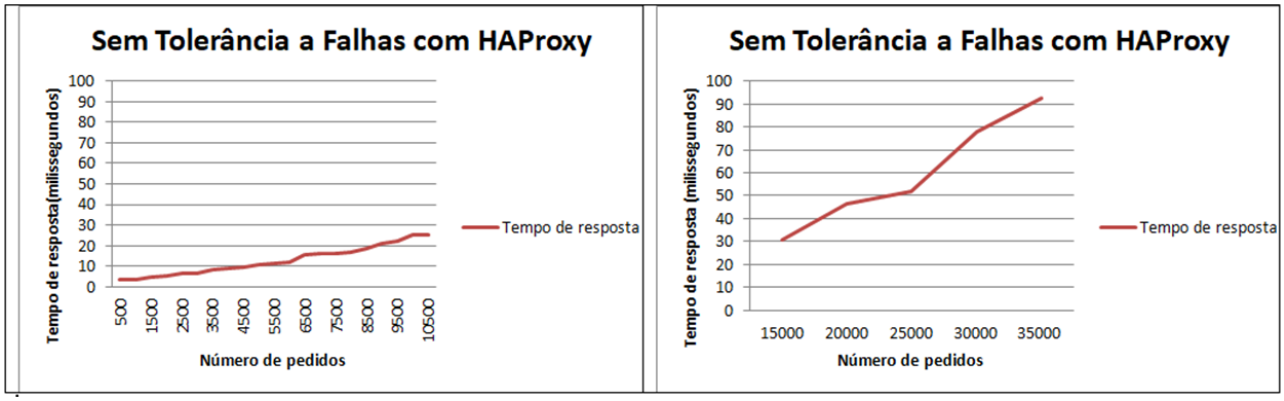

Figura 5.5: Resposta do servidor sem tolerância a falhas com HAProxy.

5.2 Avaliação do Desempenho da Aplicação Web com Elevada Disponibilidade

Nesta secção é avaliado o desempenho da aplicação na cloud, desta feita, com a réplica dos servidores . Com descrito no capítulo anterior a réplica dos servidores web é feita através do servidor HAProxy e os servidores do MySQL foram configurados para que haja réplica entre eles. Após o término da implementação e configuração do cenário de aplicação web com alta disponibilidade, os passos seguintes debruçam-se na feitura dos testes com as ferramentas de avaliação de desempenho do website. Os testes foram efetuados com o intuito de analisar o comportamento do sistema perante as situações de falhas, falhas essas que foram provocadas intencionalmente. Para tal, foram cronometrados os tempos de acesso à página por parte das máquinas dos utilizadores e da resposta dos servidores, no mesmo molde realizados nos dois cenários anteriores. Para a realização dos testes, consideram-se sete cenários: cenário sem falhas, onde são realizados os testes com os quatro servidores ativos, a seguir serão apresentados os cenários com apenas uma falha, em que pode falhar um servidor web ou um servidor de base de dados, e por último são descritas os cenários com duas falhas, isto é, falha um servidor web numa máquina virtual e um servidor de base dados noutra máquina virtual.

a) Cenário sem falhas

Neste cenário que se baseia na configuração de alta disponibilidade apresentada no capítulo anterior, as falhas dos servidores web e MySQL são toleradas, pois, se, por exemplo, o servidor web de máquina virtual 1 falhar, o outro retomará o serviço e vice-versa, também acontece a mesma coisa com os servidores MySQL. Nesta hipótese, as falhas não são toleradas se:

- Os quatro (4) servidores (web e MySQL) falharem ao mesmo tempo;
- Um servidor web estiver ligado e se estiverem desligados o outro servidor web e os dois servidores MySQL;
- Dois servidores web estiverem ligados ou os dois servidores MySQl estiverem desligados.

Foram registados os seguintes valores para as métricas do lado do cliente, 0,186 segundos (s) para o tempo de carregamento e 0,170 milissegundos (ms) para o tempo de conexão(ver tabela 5.3). Em relação aos dois cenários anteriores, o tempo de carregamento baixou ligeiramente, isto é, 0,002 segundos em relação ao cenário sem tolerância a falhas e 0,013 segundos em relação ao cenário sem tolerância a falhas com HAProxy. O tempo de conexão aumentou 0,002 milissegundos.

Tabela 5.3: Avaliação do desempenho (lado do cliente) com alta disponibilidade: cenário sem falhas .

Avaliação do desempenho (lado do cliente) com alta disponibilidade: cenário sem falhas	
Tempo de carregamento (Load Time)	Tempo de conexão (First Byte)
0,186 segundos	0,170 milissegundos

Para as métricas do lado do servidor, os valores registados estão na figura 5.6 e de forma gráfica na figura 5.7. A média aritmética do tempo de resposta é 9,1 milissegundos, levando em conta os dois cenários anteriores, a média baixou aproximadamente 29,4% milissegundos e 32,8% milissegundos em relação aos cenários da "Aplicação Exemplo" e "Aplicação Exemplo com HAProxy" respectivamente. A comparação deste cenário com os cenários anteriores, deve-se ao facto de serem cenários que têm algo em comum, isto é, são cenários sem falhas.

Número de pedidos	500	1500	2500	3500	4500	5500	6500	7500	8500	9500	10500
Tempo de resposta (milissegundos)	3,80	2,67	3,53	5,79	5,94	7,95	10,10	11,50	13,30	14,60	16,80
Tempo de resposta (milissegundos)	4,00	3,00	4,00	6,00	6,00	8,00	10,00	11,00	13,00	14,00	16,00

Figura 5.6: Avaliação do desempenho (lado do servidor) com alta disponibilidade: cenário sem falhas.

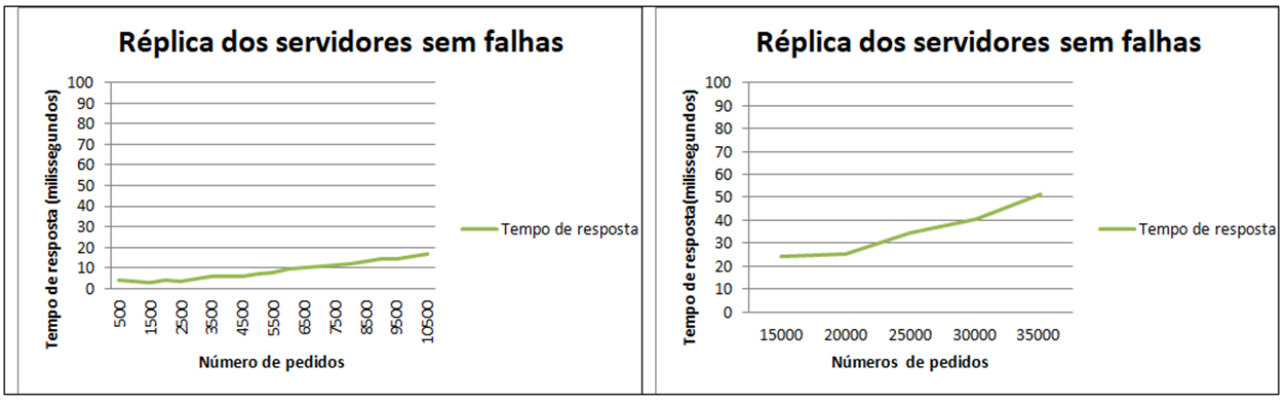

Figura 5.7: Ambiente de alta disponibilidade sem falhas.

b) Falha do servidor web da máquina virtual 1

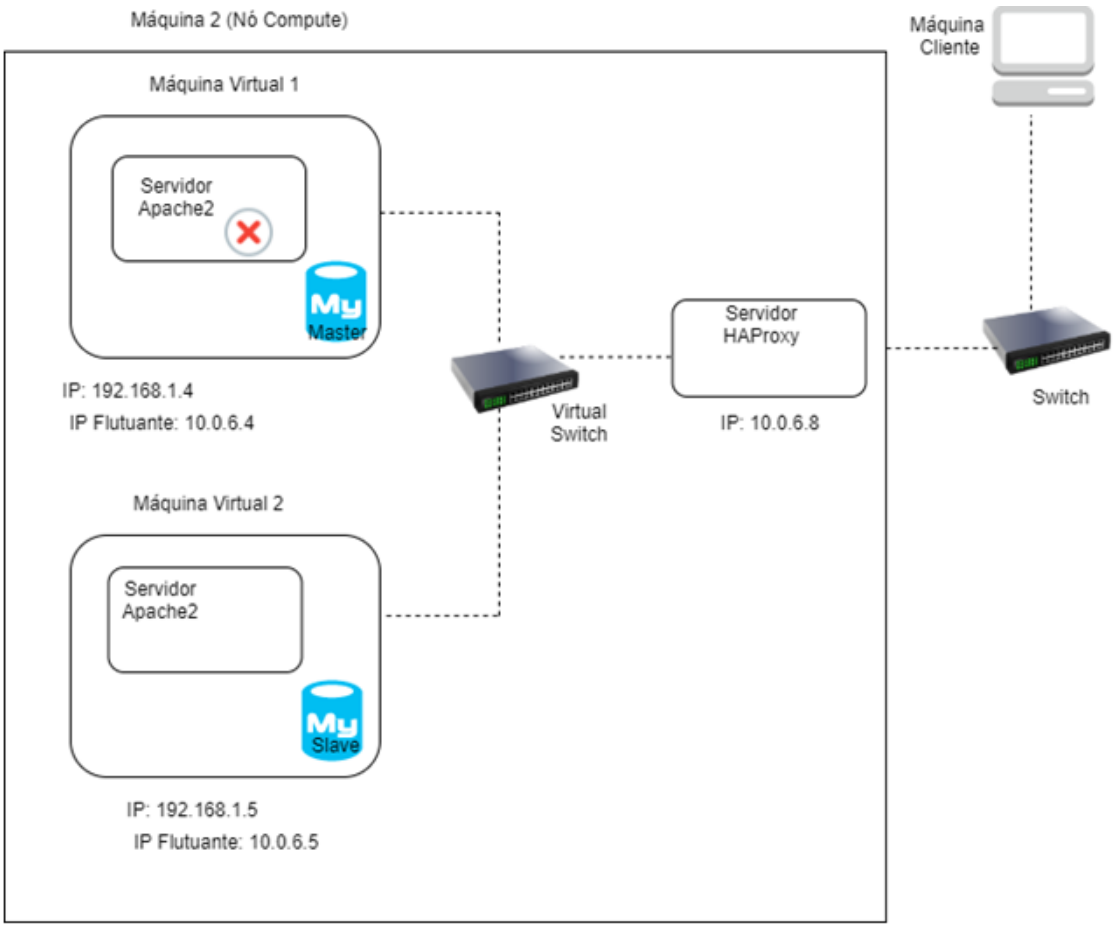

Figura 5.8: Falha do servidor web da máquina virtual 1.

Neste cenário foram cronometrados os tempos após a falha do servidor web alojado na VM 1. Observando a figura 5.8, percebe-se que, está disponível um outro servidor web em VM 2 para

retomar os serviços, quando de falha do servidor web da VM 1. Mas, se falhar o servidor web da VM 2, os serviços ficarão indisponível, pois, não há um outro servidor web para dar continuidade aos pedidos.

Os resultados, no que concerne, às métricas do lado do cliente, conforme demonstra a tabela 5.4, foram os seguintes: o tempo de carregamento (load time) foi de 0,218 segundos (s) e o First Byte (tempo de conexão) registou 0,174 milissegundos (ms). A análise do lado cliente não mostra diferenças com significado do cenário com replicação em relação ao cenário sem replicação.

Tabela 5.4: Avaliação do desempenho (lado do cliente) com alta disponibilidade: cenário com falha do servidor web da máquina virtual 1 .

Avaliação do desempenho (lado do cliente) com alta disponibilidade: cenário com falha do servidor web da VM1	
Tempo de carregamento (Load Time)	Tempo de conexão (First Byte)
0,218 segundos	0,174 milissegundos

Os valores em relação às métricas do lado do servidor estão na figura 5.9, em que a média aritmética do tempo de resposta é 12,83 milissegundos conjunto de valores já referidos.

Número de pedidos	500	1500	2500	3500	4500	5500	6500	7500	8500	9500	10500
Tempo de resposta (milissegundos)	3,40	4,94	6,96	8,18	10,10	11,90	13,60	15,90	18,30	20,30	22,70
Tempo de resposta (milissegundos)	2,00	5,00	7,00	8,00	10,00	12,00	13,00	16,00	18,00	20,00	22,00

Figura 5.9: Avaliação do desempenho (lado do servidor) com alta disponibilidade: cenário com falhas do servidor web da máquina virtual 1.

A figura 5.10 demonstra graficamente os resultados dos dois conjuntos dos testes.

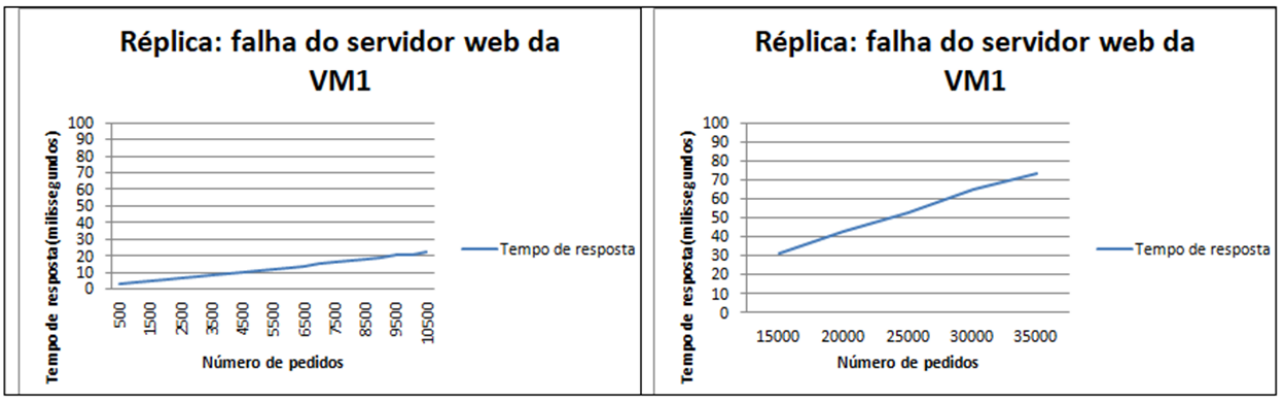

Figura 5.10: Ambiente da alta disponibilidade (falha do servidor web da VM 1).

c) Falha do servidor web da máquina virtual 2

Este cenário difere do cenário anterior (alínea b), pois, neste caso, há falha do servidor web da VM 2, conforme ilustra a figura 5.11.

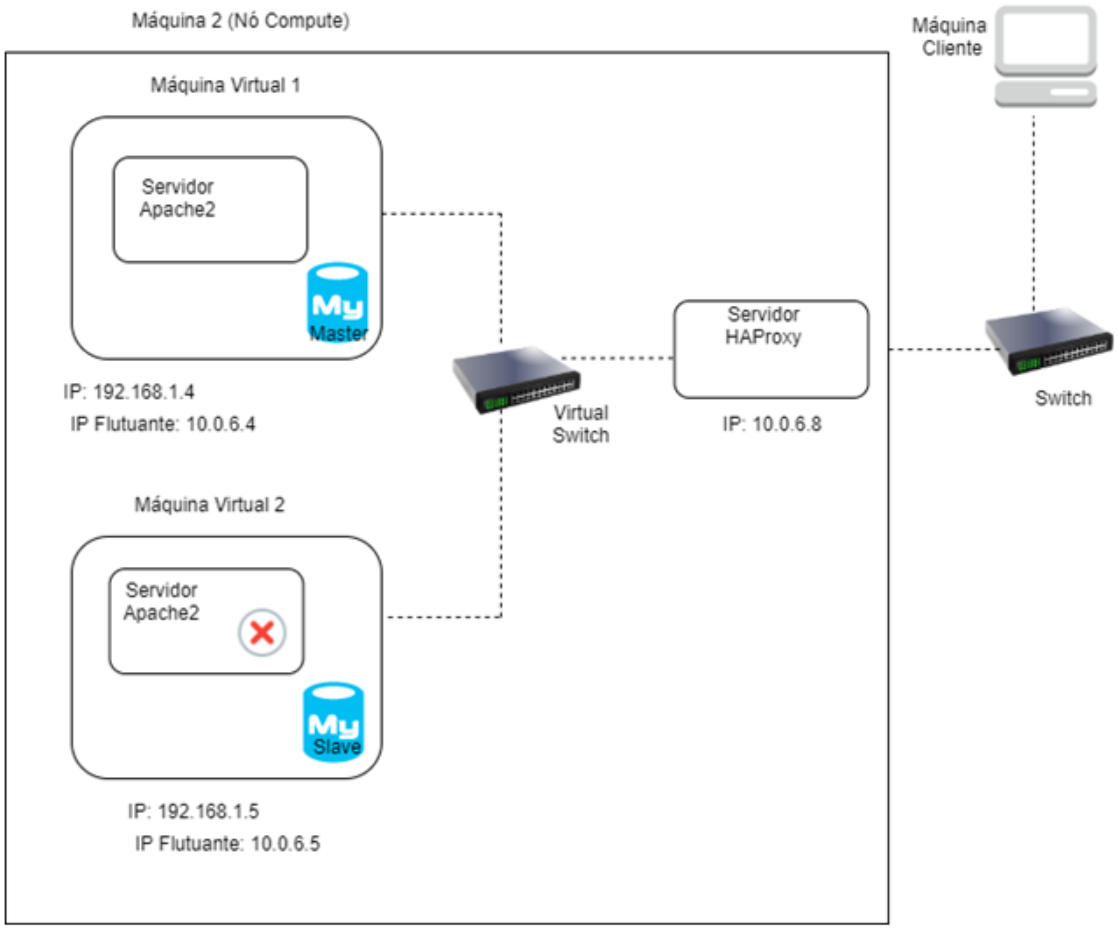

Figura 5.11: Falha do servidor web da máquina virtual 2.

Os resultados no que concerne às métricas do lado do cliente foram os seguintes: o tempo de carregamento (load time) foi de 0,190 segundos (s) e o First Byte (tempo de conexão) registou 0,172 milissegundos (ms), conforme demonstra a tabela 5.5.

Tabela 5.5: Avaliação do desempenho (lado do cliente) com alta disponibilidade:cenário com falha do servidor web da máquina virtual 2 .

Avaliação do desempenho (lado do cliente) com alta disponibilidade: cenário com falha do servidor web da VM2	
Tempo de carregamento (Load Time)	Tempo de conexão (First Byte)
0,190 segundos	0,172 milissegundos

Em relação às métricas do lado do servidor, os valores obtidos estão na figura 5.12. Desta feita, a média aritmética do tempo de resposta, comparando com o cenário anterior (alínea b) baixou 18,39%, ou seja, a média aritmética é 10,47 milissegundos.

Número de pedidos	500	1500	2500	3500	4500	5500	6500	7500	8500	9500	10500
Tempo de resposta (milissegundos)	2,97	3,77	4,90	7,18	8,33	9,61	11,40	13,20	14,70	16,60	18,40
Tempo de resposta (milissegundos)	3,00	4,00	5,00	7,00	8,00	10,00	11,00	13,00	15,00	16,00	18,00

Figura 5.12: Avaliação do desempenho (lado do servidor) com alta disponibilidade: cenário com falhas do servidor web da máquina virtual 2.

Os resultados sob a forma gráfica apresentam-se na figura 5.13.

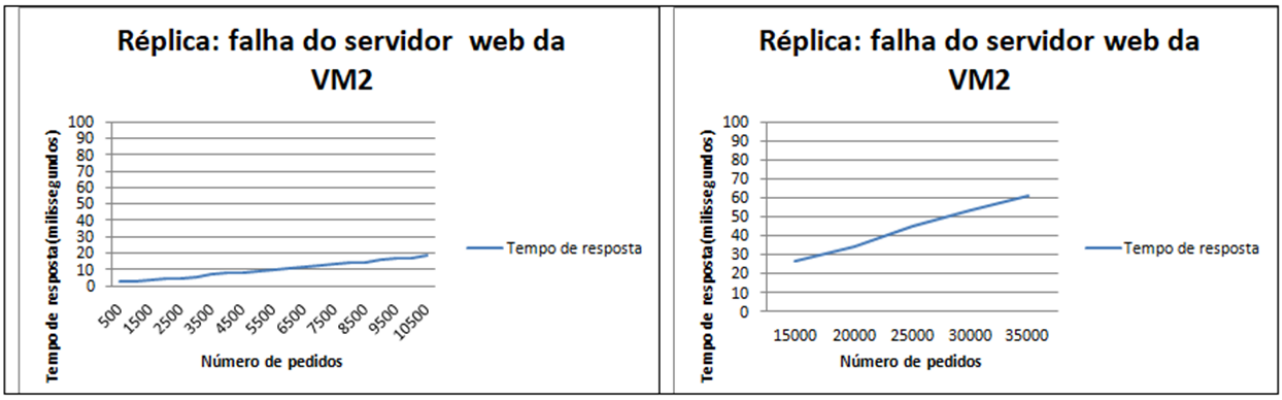

Figura 5.13: Ambiente da alta disponibilidade (falha do servidor web da VM 2).

d) Falha do servidor MySQL "Slave" da máquina virtual 2

Neste caso, admite-se tolerância a falhas só nos servidores web, isto é, se o servidor web alojado na máquina virtual 1 falhar, o servidor web alojado na máquina virtual 2 retomará o serviço, ou vice-versa. Todavia, não tolera a falha no servidor MySQL, visto que o servidor MySQL da máquina virtual 2 ou do nó "Slave" está desativado (ver figura 5.14).

Nesta hipótese, como se pode ver na tabela 5.6, o tempo de carregamento foi de 0,192 segundos (s) e o tempo de conexão registou 0,162 milissegundos (ms), isto em relação às métricas do lado do cliente.

Tabela 5.6: Avaliação do desempenho (lado do cliente) com alta disponibilidade: cenário com falha do MySQL da máquina virtual 2 .

Avaliação do desempenho (lado do cliente) com alta disponibilidade: cenário com falha do MySQL da VM2	
Tempo de carregamento (Load Time)	Tempo de conexão (First Byte)
0,192 segundos	0,162 milissegundos

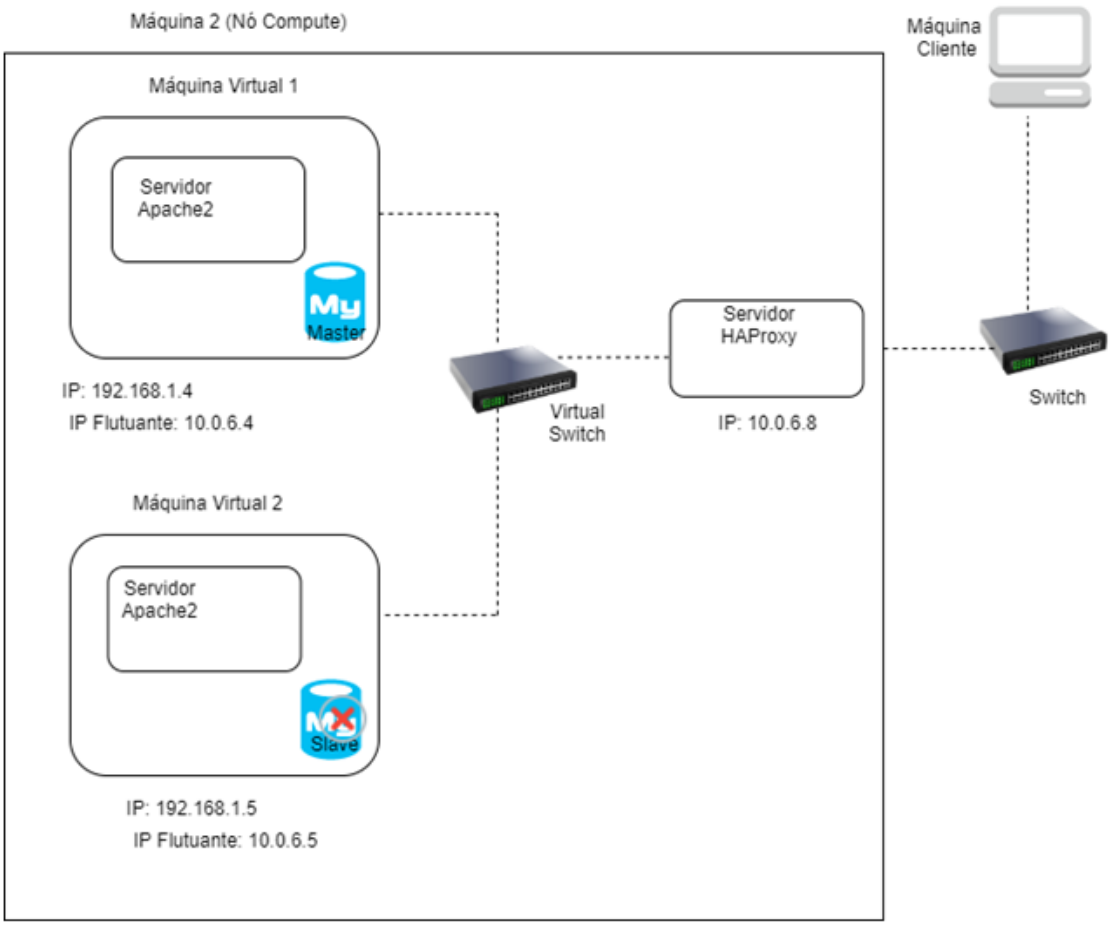

Figura 5.14: Falha do servidor MySQL "Slave" da máquina virtual 2.

Os resultados do lado do servidor, no que tange, ao tempo de resposta e tempo de conexão dos pedidos solicitados ao servidores são demonstrados na figura 5.15. Neste cenário, a média aritmética do tempo de resposta é 6,97 milissegundos

Número de pedidos	500	1500	2500	3500	4500	5500	6500	7500	8500	9500	10500
Tempo de resposta (milissegundos)	2,933	2,359	3,513	4,091	4,865	6,043	6,848	8,6	10,1	11,4	12,8
Tempo de conexão (milissegundos)	3	2	3	4	5	6	7	8	10	11	13

Figura 5.15: Avaliação do desempenho (lado do servidor) com alta disponibilidade: cenário com falhas do servidor MySQL da máquina virtual 2.

A relação entre o número de pedidos e tempo de resposta retratados para os dois intervalos estudados é demonstrado graficamente através da figura 5.16.

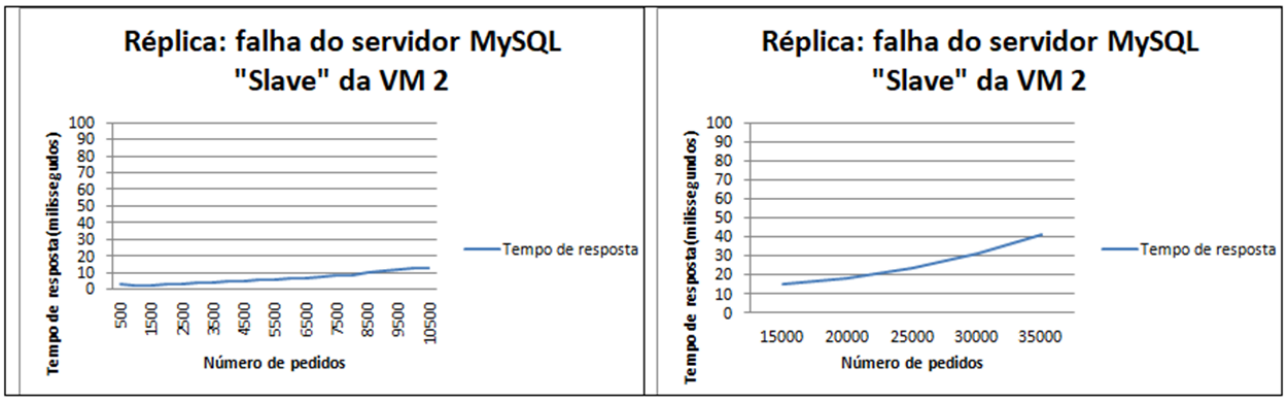

Figura 5.16: Ambiente da alta disponibilidade (falha do servidor MySQL “Slave” da máquina virtual 2).

e) Falha do servidor MySQL “Master” da máquina virtual 1

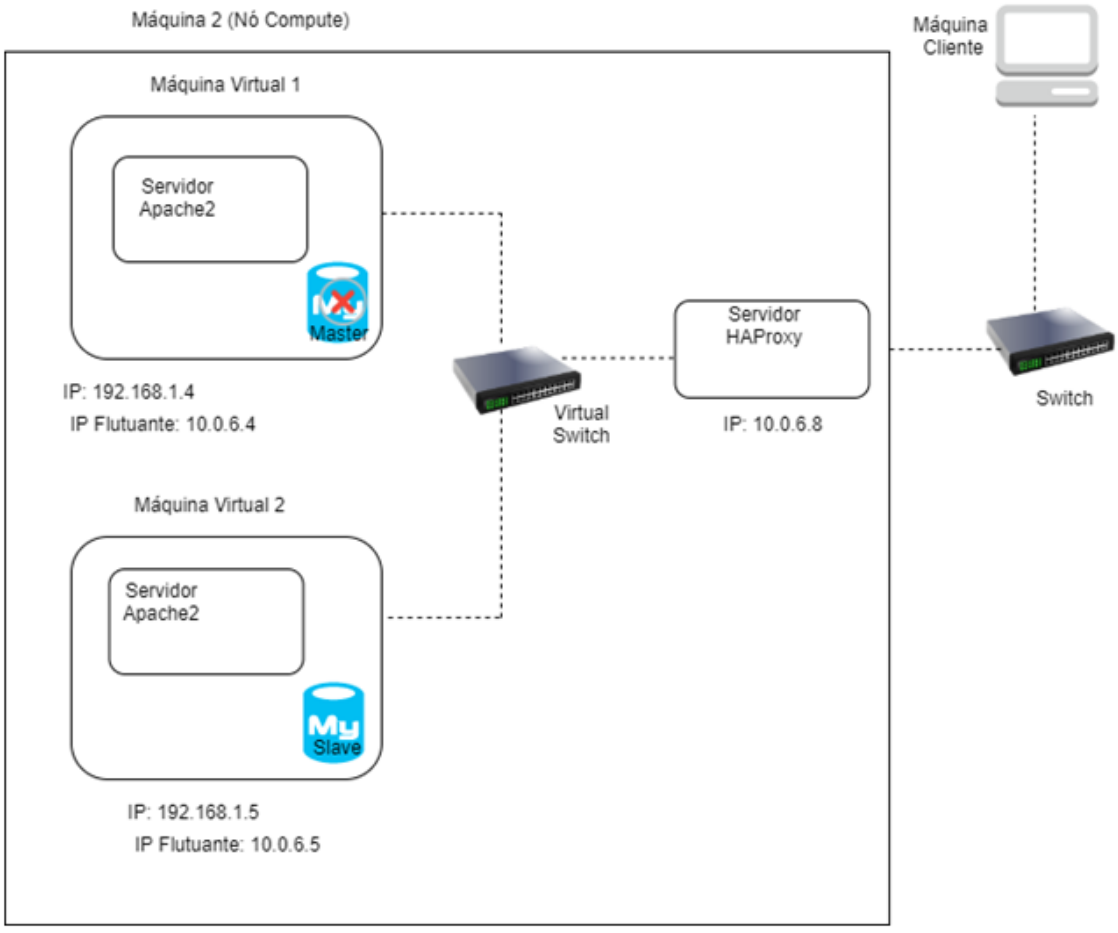

Figura 5.17: Falha do servidor MySQL “Master” da máquina virtual 1.

Neste cenário a falha é no servidor MySQl “Master” da máquina virtual 1 (ver figura 5.17).

Nesta hipótese, como se pode ver na tabela 5.7, o tempo de carregamento foi de 0,184 segundos (s) e o tempo de conexão registou 0,168 milissegundos (ms), isto é, em relação às métricas do lado do cliente.

Tabela 5.7: Avaliação do desempenho (lado do cliente) com alta disponibilidade: cenário com falha do MySQL da máquina virtual 1.

Avaliação do desempenho (lado do cliente) com alta disponibilidade: cenário com falha do MySQL da VM1	
Tempo de carregamento (Load Time)	Tempo de conexão (First Byte)
0,184 segundos	0,168 milissegundos

Em relação às métricas do lado do servidor, os resultados são ilustrados na figura 5.18. A média aritmética do tempo de resposta, registou um valor ligeiramente superior, em relação ao cenário anterior (alínea d), o valor registado foi de 7,92 milissegundos, isto é, subiu 13,6%. A representação gráfica apresenta-se na figura 5.19

Número de pedidos	500	1500	2500	3500	4500	5500	6500	7500	8500	9500	10500
Tempo de resposta (milissegundos)	3,08	2,72	3,44	4,57	5,85	6,29	9,37	10,10	11,90	12,80	14,00
Tempo de conexão (milissegundos)	3,00	3,00	3,00	5,00	6,00	6,00	9,00	10,00	12,00	13,00	14,00

Figura 5.18: Avaliação do desempenho (lado do servidor) com alta disponibilidade: cenário com falhas do servidor MySQL da máquina virtual 1.

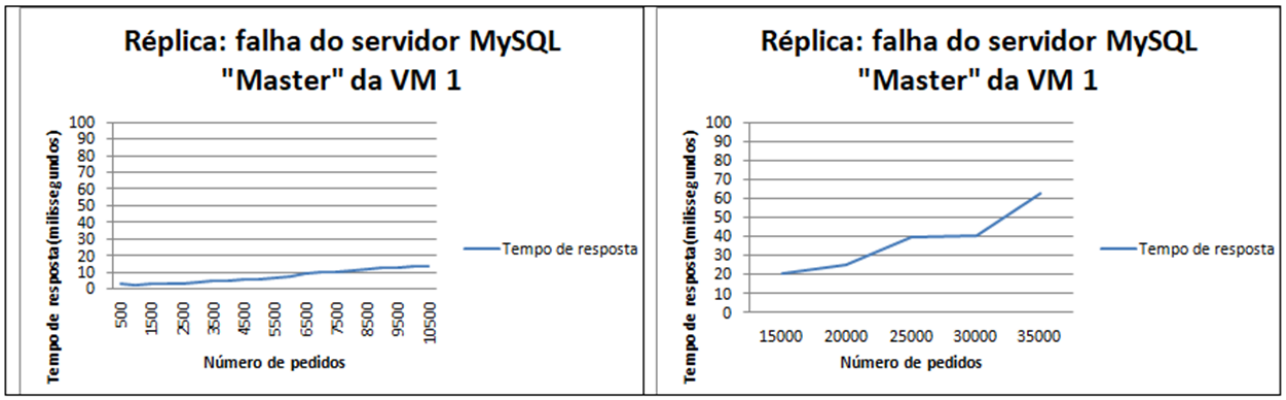

Figura 5.19: Ambiente da alta disponibilidade (falha do servidor MySQL "Master" da máquina virtual 1).

f) Falha do servidor web da máquina virtual 1 e do servidor MySQL "Slave" da máquina virtual 2

Neste cenário, dois servidores, um web e MySQL "Slave" alojados em máquinas virtuais diferentes falham ao mesmo tempo (ver figura 5.20). Mas, mesmo assim a aplicação continua

disponível, pois, em cada máquina virtual tem um servidor disponível para retomar o serviço por causa das falhas dos servidores mencionados.

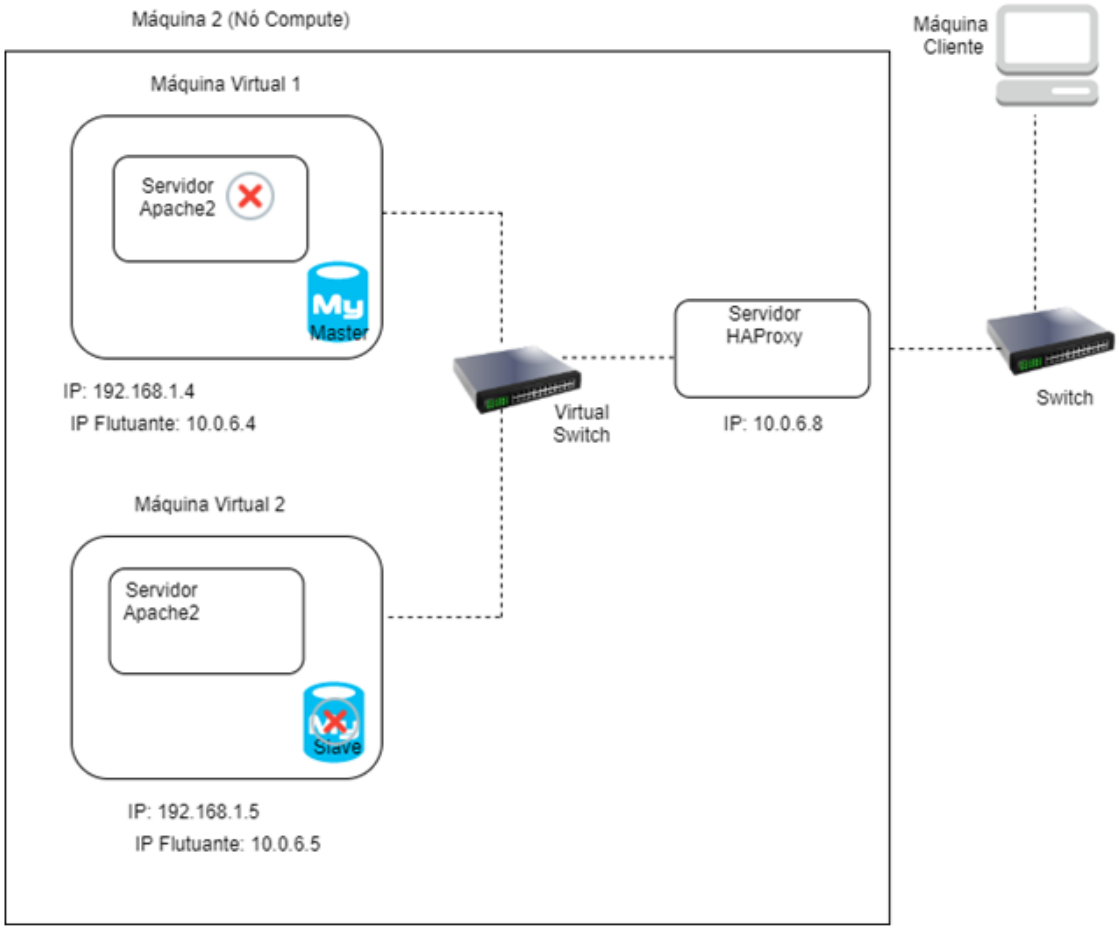

Figura 5.20: Falha do servidor web da VM1 e do servidor MySQL “Slave” da VM2.

Nesta hipótese, como se pode ver na tabela 5.8, o tempo de carregamento foi de 0,195 segundos (s) e o tempo de conexão registou 0,167 milissegundos (ms), isto em relação às métricas do lado de cliente.

Tabela 5.8: Avaliação do desempenho (lado do cliente) com alta disponibilidade: cenário com falha do servidor web da máquina virtual 1 e do servidor MySQL da máquina virtual 2.

Avaliação do desempenho (lado do cliente) com alta disponibilidade: cenário com falha do servidor web da VM1 e do servidor MySQL da VM2	
Tempo de carregamento (Load Time)	Tempo de conexão (First Byte)
0,195 segundos	0,167 milissegundos

Os resultados dos testes do lado do servidor são demonstrados na figura 5.21. A média aritmética do tempo de resposta é 5,6 milissegundos.

O gráfico ilustrado figura 5.22, demonstra a relação existente o número de pedidos e seu respectivo tempo de resposta.

Número de pedidos	500	1500	2500	3500	4500	5500	6500	7500	8500	9500	10500
Tempo de resposta (milissegundos)	2,46	1,38	2,24	3,40	3,53	5,61	6,23	7,70	8,34	9,28	10,40
Tempo de conexão (milissegundos)	2,00	1,00	2,00	3,00	3,00	6,00	6,00	8,00	8,00	9,00	10,00

Figura 5.21: Avaliação do desempenho (lado do servidor) com alta disponibilidade: cenário com falha do servidor web da máquina virtual 1 e do servidor MySQL da máquina virtual 2.

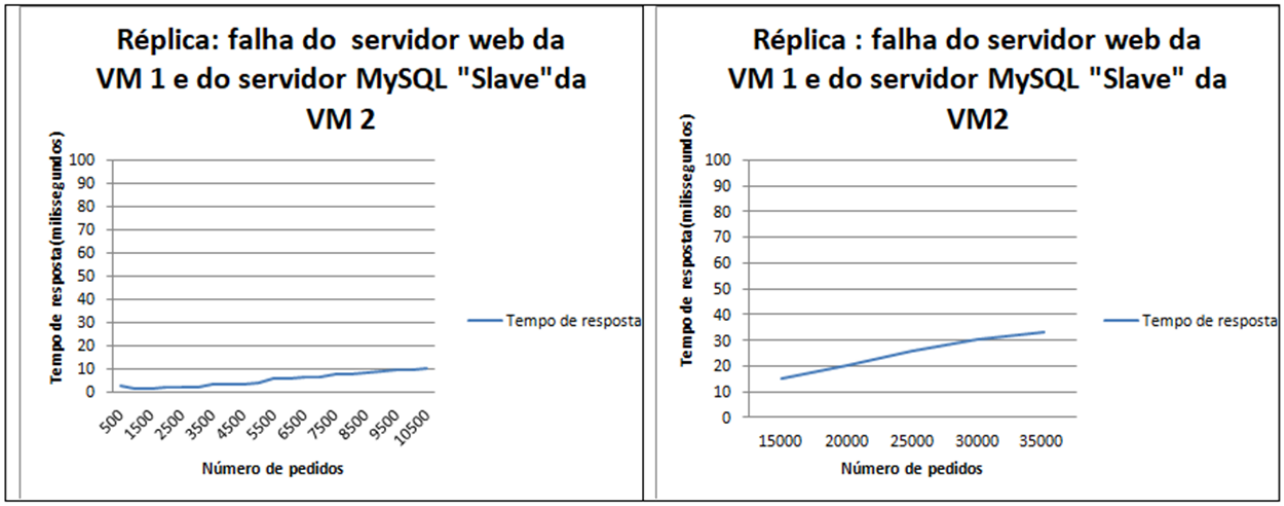

Figura 5.22: Ambiente da alta disponibilidade (falha do servidor web da VM 1 e do servidor MySQl ”Slave” da VM 2).

g) Falha do servidor web da máquina virtual 2 e do servidor MySQL “Master” da máquina virtual 1

Este cenário também tem duas falhas dos servidores (figura 5.23), mas agora falham o servidor web da VM2 e o servidor MySQl da Vm1(”Master”).

Do lado do cliente, de acordo com a tabela 5.9, registaram-se os seguintes valores, o tempo de carregamento foi de 0,187 segundos (s) e o tempo de conexão registou 0,169 milissegundos (ms).

Tabela 5.9: Avaliação do desempenho (lado do cliente) com alta disponibilidade: cenário com falha do servidor web da máquina virtual 2 e do servidor MySQL ”Master” da máquina virtual 1.

Avaliação do desempenho (lado do cliente) com alta disponibilidade: cenário com falha do servidor web da VM2 e do servidor MySQL da VM1	
Tempo de carregamento (Load Time)	Tempo de conexão (First Byte)
0,187 segundos	0,169 milissegundos

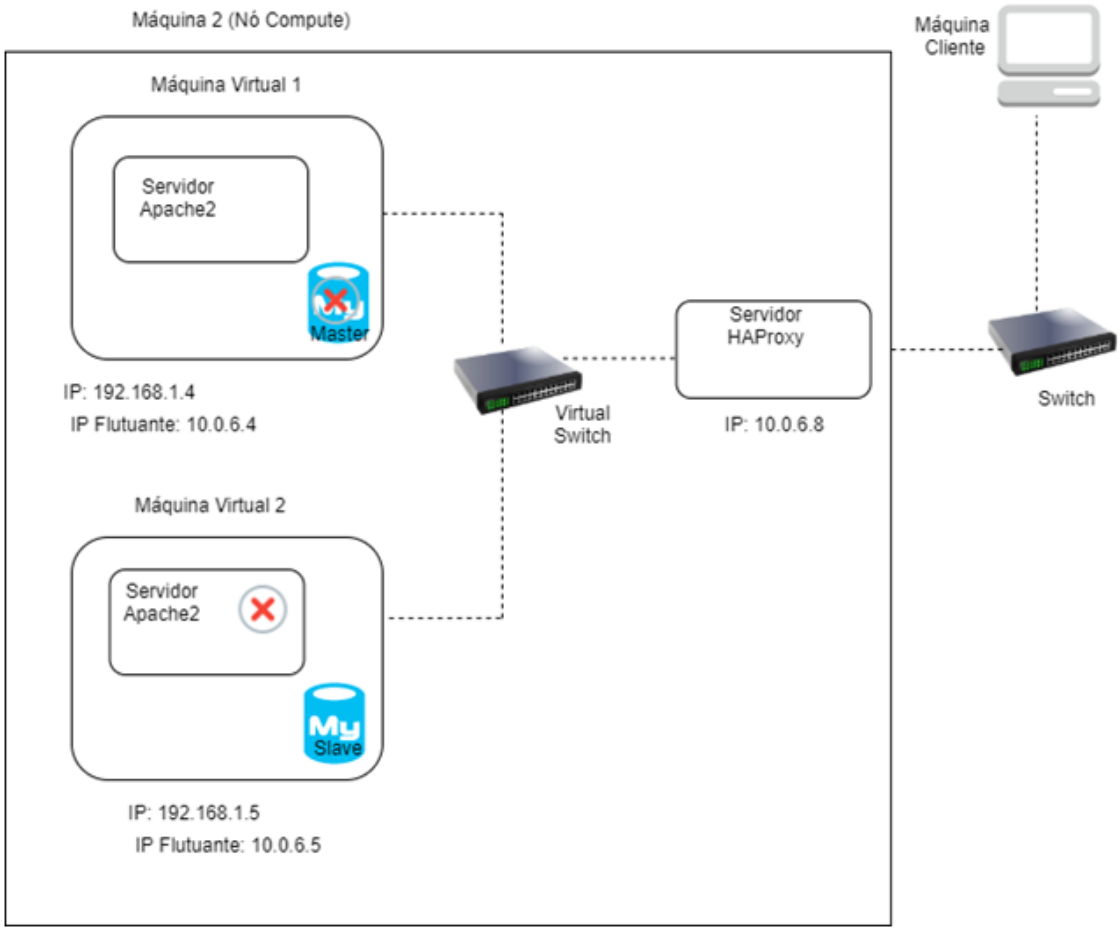

Figura 5.23: - Falha do servidor web da máquina virtual 2 e do servidor MySQL "Master" da máquina virtual 1.

Os resultados obtidos do lado do servidor estão na figura 5.24, em que a média aritmética do tempo de resposta é 5,3 milissegundos, baixou 5,4% em relação ao cenário anterior (alínea f). Os resultados sob a forma gráfica apresentam-se na figura 5.25

Número de pedidos	500	1500	2500	3500	4500	5500	6500	7500	8500	9500	10500
Tempo de resposta (milissegundos)	2,31	1,41	1,88	3,02	3,49	4,98	5,86	7,31	7,55	8,21	10,40
Tempo de conexão (milissegundos)	2,00	1,00	2,00	3,00	3,00	5,00	6,00	7,00	7,00	8,00	10,00

Figura 5.24: Avaliação do desempenho (lado do servidor) com alta disponibilidade: cenário com falha do servidor web da máquina virtual 2 e do servidor MySQL da máquina virtual 1.

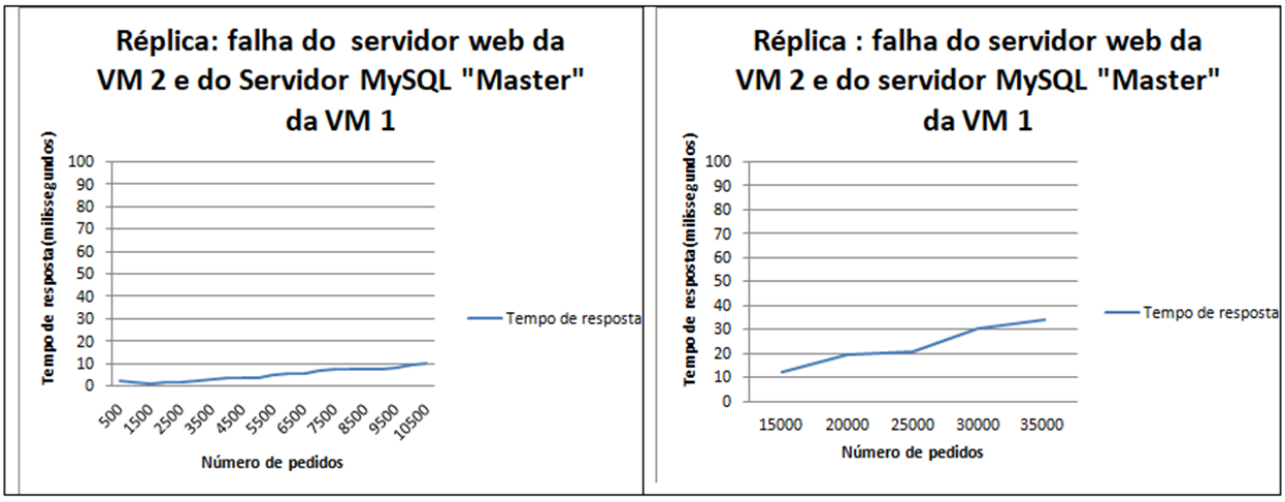

Figura 5.25: Ambiente da alta disponibilidade(falha do servidor web da VM 2 e do servidor MySQl "Master" da VM 1).

5.3 Análise da Avaliação dos Cenários

Nesta secção são analisados os resultados obtidos nos testes efetuados. Os testes podem ser divididos em três grupos:

- Situação sem falhas que engloba os cenários: sem tolerância a falhas, sem tolerância a falhas com HAproxy e com réplica dos servidores sem falhas;
- Réplica dos servidores com apenas uma falha;
- Réplica dos servidores com duas falhas.

Tabela 5.10: Resultados obtidos dos testes efetuados com a ferramenta do desempenho do website (lado do servidor).

Resultados obtidos dos testes efetuados com a ferramenta do desempenho do website (lado do servidor)	
Cenários dos testes	Tempo de resposta (média aritmética)
Sem tolerância a falhas	12,9 ms
Sem tolerância a falha - com HAProxy	13,55 ms
Réplica dos servidores	9,1 ms
Réplica - falha servidor web da VM1	12,83 ms
Réplica - falha servidor web da VM2	9,7 ms
Réplica - falha servidor MySQL "Master" da VM1	7,92 ms
Réplica - falha servidor MySQL "Slave" da VM2	6,97 ms
Réplica - falha do servidor web da VM1 e servidor MySQL "Slave" da VM2	5,6 ms
Réplica - falha do servidor web da VM2 e servidor MySQL "Master" da VM1	5,3 ms

Os resultados do lado do servidor são sumarizados na tabela 5.10 , como se pode observar qualquer dos cenários com replicação, com e sem falhas tem um tempo médio de resposta inferior aos cenários sem replicação, isto é, sem tolerância a falhas.Isto resulta do balanceamento de carga executado pelo HAproxy que permite dividir os pedidos pelas duas aplicações. Assim, apesar de introduzimos um ponto único de falha há vantagem na replicação pois temos um sistema mais rápido e em que a falha de alguns compenentes é tolerada. O HAProxy apesar de ser um ponto único de falha corresponde a uma fracção de computação muito pequena em relação aos outros componentes dos sistema , servidor web e MySQL, logo a probalidade de falhar é menor.

De seguida apresentam-se os gráficos de tempo resposta em função do número de pedidos para os três cenários sem falhas (aplicação exemplo, aplicação exemplo com HAProxy e aplicação com alta disponibilidade), figura 5.26, para os quatro cenários com uma falha , figura 5.27 e para os dois cenários com dua falhas , figura 5.28.

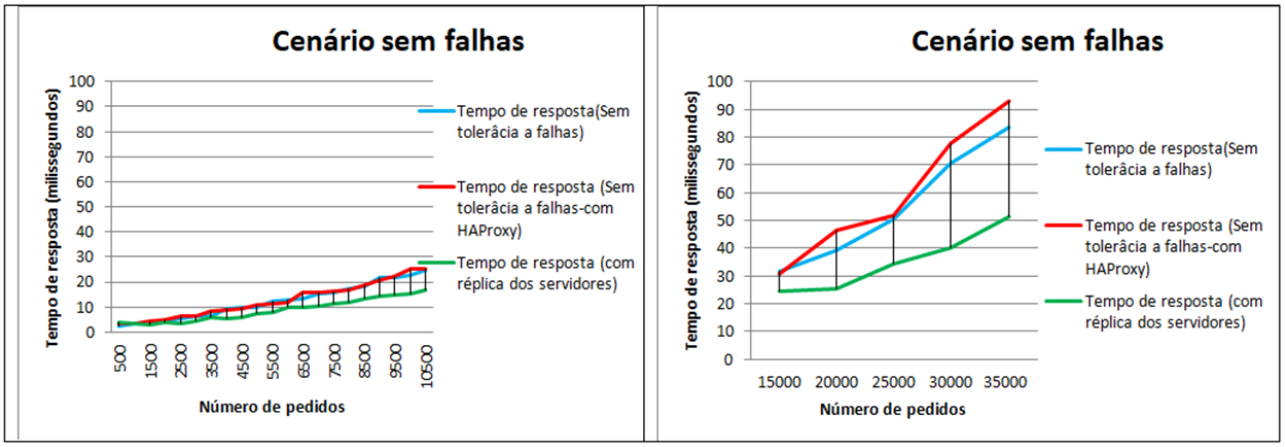

Figura 5.26: Cenários sem falhas.

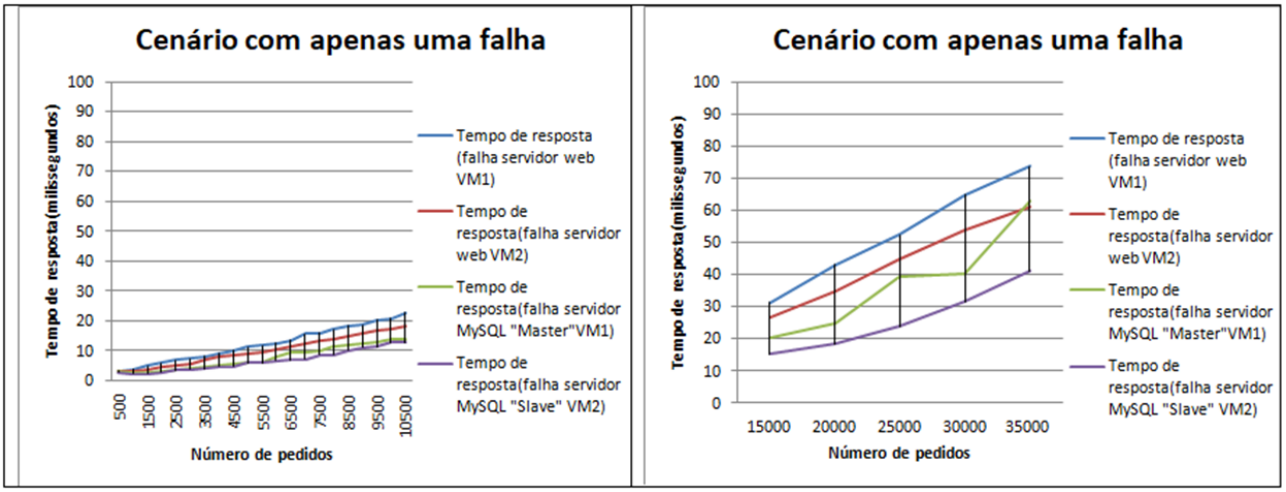

Figura 5.27: Cenários com apenas uma falha.

Como se pode observar, nos cenários com falhas o melhor tempo de resposta ocorre quando falha o servidor MySQL. Este resultado parece indicar que a replicação de dados tem um custo superior à replicação dos processos.

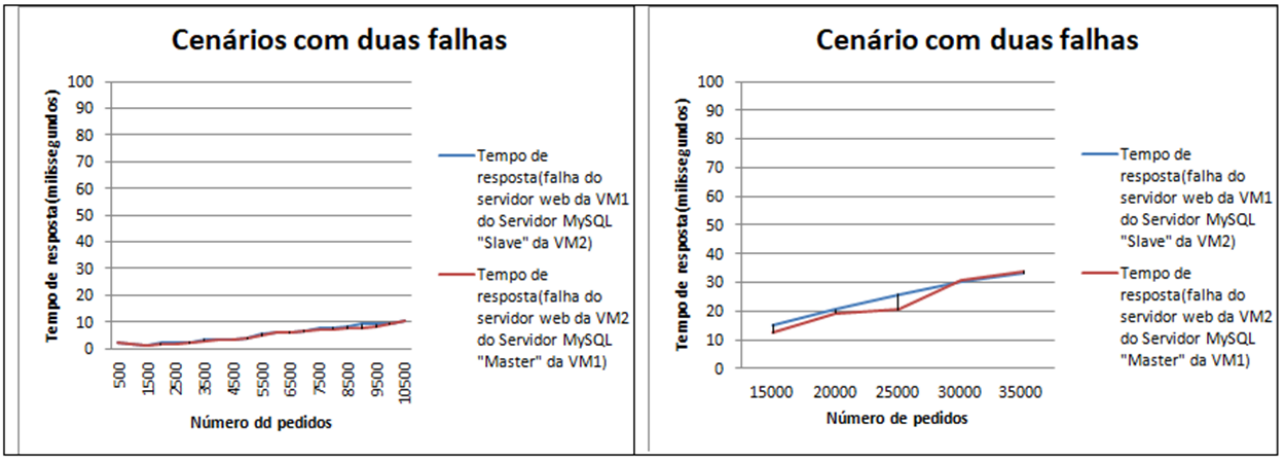

Figura 5.28: Cenários com duas falhas.

5.4 Conclusão

Após a realização dos testes de avaliação do desempenho foi possível constatar que um sistema com alta disponibilidade (balanceamento de carga e mecanismo de réplica) é uma solução viável para servidores alojados nas máquinas virtuais instanciadas num sistema da cloud, visto que é garantida a melhoria do desempenho dos servidores e oferece tolerância a falhas de alguns componentes da aplicação.

Capítulo 6

Conclusão e Trabalho Futuro

Este trabalho permitiu consolidar os conceitos explanados no estado da arte, nomeadamente computação na cloud, tolerância a falhas e alta disponibilidade. Com a realização deste projeto, foi possível entender, de uma forma pormenorizada, a instalação e o funcionamento do OpenStack, isto é, a função de cada nó e a relação entre eles, a relação entre os serviços existentes. Ainda possibilitou instanciar máquinas virtuais, configuração das suas redes internas e externas e atribui-lhes IPs flutuantes. Em suma, concluiu-se que a plataforma cloud OpenStack é um sistema de cloud maduro, apesar de ser Open source. Também tem um enorme potencial de evolução, visto que em cada seis meses é lançada uma nova versão.

Um outro ganho, que este trabalho proporcionou, foi dar a conhecer como configurar um sistema para alta disponibilidade (balanceamento de carga e réplica) nos servidores alojados na infraestrutura cloud Openstack. Para verificar, se é viável ou não ter sistemas com alta disponibilidade nos servidores alojados no sistema da cloud, foram realizados vários testes para diferentes cenários, com o intuito de avaliar o tempo de execução e desempenho, por fim concluiu-se, que o cenário com réplica dos servidores é um cenário viável, pois, garante a disponibilidade dos serviços, mesmo em caso da falha destes, além da eventual vantagem de melhorar o desempenho.

Como é normal na execução de qualquer projeto, sempre surgem obstáculos de diversas ordens e este trabalho não é exceção. A primeira limitação encontrada na implementação da plataforma OpenStack foi a quantidade de máquinas físicas, isto é, três (3) máquinas e, consequentemente, três nós, mesmo assim atendeu ao requisito mínimo para a instalação do OpenStack. Uma outra limitação encontrada foi a configuração das redes nas máquinas virtuais, visto que a rede física é privada. Mas, com muita dedicação e pesquisa este problema foi solucionado.

Este trabalho servirá de alavanca para novas pesquisas sobre a infraestrutura da cloud OpenStack, nomeadamente no capítulo de tolerância a falhas. Sabendo que a computação na cloud e, em particular, o OpenStack é um marco recente na área da computação, ainda tem muito para se pesquisar, neste contexto, como trabalho futuro, a sugestão é desenvolver um mecanismo que tolera as falhas nos nós e serviços da cloud OpenStack.

Referências

[1] M. Sharma, «Cloud Computing Architecture Services,» *IOSR J. Comput. Eng.*, vol. 19, n.º 2, pp. 13–18, 2017.

[2] A. Rashid e A. Chaturvedi, «Cloud Computing Characteristics and Services: A Brief Review,» *International Journal of Computer Sciences and Engineering Open Access Cloud Computing Characteristics and Services*, vol. 7, n.º 1, pp. 1–7, March ,2019.

[3] C. Taurion, *Cloud Cumputing :Computação em Nuvem Transformando o mundo da Tecnologia da Informação*. Rio de Janeiro: Brasport, 2009.

[4] P. Mell e T. Grance, «The nist definition of cloud computing. recommendations of the national institute of standards and technology,» *NIST Special Publication*, vol. 145, n.º 6, pp. 1–2, 2011.

[5] S. S. Madani e S. Jamali, «A Comparative Study of Fault Tolerance Techniques in Cloud Computing,» *International Journal of Research in Computer Applications and Robotics*, vol. 6, n.º 3, pp. 7–15, 2018.

[6] M. Armbrust e M. Zaharia, «A view of cloud computing,» *Commun. ACM*, vol. 53, n.º 4, pp. 50–58, 2010.

[7] L. B. A. Tchana e D. Hagimont, «Fault Tolerant Approaches in Cloud Computing Infrastructures,» *The Eighth International Conference on Autonomic and Autonomous Systems ICAS*, n.º 12, pp. 42–48, 2012.

[8] J. Kaur e S. Kinger, «Analysis of Different Techniques Used For Fault Tolerance,» *Int. J. Comput. Sci. Inf. Technol*, vol. 5, n.º 3, pp. 4086–4090, 2016.

[9] R. Gate. (2019). Cloud Computing Service Model. accessed 02-27-2019, URL: `https://www.researchgate.net`.

[10] G. C. F. e J. J. D. K. Hwang, *Distributed and Cloud-From Parallel Processing to the Internet of Things*. Waltham: USA:Morgan Kaufmann, 2012.

[11] Microsoft. (2018). O que é Computação em nuvem. accessed 02-05-2018, URL: `https://azure.microsoft.com/pt-br/overview/what-is-cloud-computing/`.

[12] OpenStack. (2017). What is OpenStack. accessed 02-06-2018, URL: `https://www.openstack.org/`.

[13] NASA. (2015). Nebula,NASA, and OpenStack. accessed 02-09-2018, URL: `https://open.nasa.gov/blog/nebula-nasa-and-openstack/`.

[14] OpenStack. (2017). What is OpenStack. accessed 02-10-2018, URL: `https://docs.openstack.org/mitaka/install-guide-ubuntu/overview.html`.

[15] AWS. (2018). Centro de arquitetura da AWS. accessed 02-11-2018, URL: `https://aws.amazon.com/pt/architecture/`.

[16] ——, (2018). Amazon EC2. accessed 02-11-2018, URL: `https://aws.amazon.com/pt/ec2/`.

[17] ——, (2019). AWS Lambda. accessed 02-03-2019, URL: `https://aws.amazon.com`.

[18] ——, (2019). Amazon Elastic Block Store. accessed 02-03-2019, URL: `https://aws.amazon.com`.

[19] P. J. P. da Costa e A. M. R. da Cruz, «Migration to Windows Azure – Analysis and Comparison,» *Procedia Technol*, vol. 5, pp. 93–102, 2012.

[20] Microsoft. (2018). O que é uma rede de entrega de conteúdos (CND) no Azure. accessed 02-11-2018, URL: https://docs.microsoft.com/pt-pt/azure/cdn/cdn-overview.

[21] M. P. Ltd. (2008-2017). Aneka overview and architecture. accessed 02-11-2018, URL: www.manjrasoft.com.

[22] M. Portnoy, *Virtualization Essentials*. John Wiley Sons, Inc., Indianapolis, Indiana, 2012.

[23] Wikipédia. (2019). Virtualização. accessed 02-03-2019, URL: https://pt.wikipedia.org/wiki/Virtualiza%C3%A7%C3%A3o.

[24] Dunamys. (2016-2018). Virtualização-Definição, Tipos e Benefícios. accessed 02-03-2019, URL: https://www.dunamys.inf.br.

[25] Microsoft. (2019). O que são as máquinas virtuais. accessed 02-03-2019, URL: https://azure.microsoft.com/pt-pt/overview/what-is-a-virtual-machine/.

[26] R. Hertzog e R.Mas. (2019). The Debian Administrator's Handbook. accessed 02-04-2019, URL: https://debian-handbook.info.

[27] H. P. MARTINS, «Tolerância a falha em um ambiente de computação em nuvem open source,» tese de mestrado, Universidade Estadual Paulista Julio de Mesquita Filho, Instituto de Biociências, Letras e Ciências Exatas, São Paulo, 2014, p. 82.

[28] InfoWester. (2001-2019). O que é virtualização e para que serve. accessed 02-04-2019, URL: https://www.infowester.com/virtualizacao.

[29] B. Emer, «Implementação de alta disponibilidade em uma empresa Prestadora de serviços para Internet,» tese de mestrado, Universidade de Caxias Do SulCentro de Ciências Exatas e da Tecnologia Ciência da Computação, Caxias do Sul, 2016, p. 81.

[30] A. Lima. (2010). Tolerância a Falhas em Sistemas Distribuídos. accessed 02-11-2018, URL: https://andreyhlb.files.wordpress.com.

[31] F. de Engenharia da Universidade do Porto. (2010). Tolerância a Falhas. accessed 02-15-2018, URL: https://paginas.fe.up.pt/~pfs/aulas/sd2010/at/23ft.pdf.

[32] S. Suguna e K. Devi, «VMFT: VIRTUAL MACHINE FAULT TOLERANCE IN CLOUD COMPUTING,» vol. 22, n.º 2, pp. 256–265, 2016.

[33] A. Bala e I. Chana, «Fault Tolerance- Challenges , Techniques and Implementation in Cloud Computing,» *Int. J. Comput. Sci.*, vol. 9, n.º 1, pp. 288–293, 2012.

[34] C. L. B. Randell e C. Landwehr, «Basic Concepts and Taxonomy of Dependable and Secure Computing,» *IEEE Transactions on Dependable and Secure Computing*, vol. 1, n.º 1, January-March 2004.

[35] H. Agarwal e A. Sharma, «A comprehensive survey of Fault Tolerance techniques in Cloud Computing,» *International Conference on Computing and Network Communications (CoCoNet), Trivandrum*, n.º 408-413, 2015.

[36] T. S. Weber, «Um roteiro para exploração dos conceitos básicos de tolerância a falhas,» *Curso Espec. em Redes e Sist. Distrib. - Inst. Informática - UFRGS*, pp. 1–62, 2002.

[37] D. Clinton. (2017). High Availability: Concepts and Theory. accessed 02-04-2019, URL: https://hackernoon.com/high-availability.

[38] OpenStack. (2018). Introduction to high availability. accessed 02-04-2019, URL: https://docs.openstack.org/ha-guide/intro-ha.html.

[39] H. Kaur e A. Kaur, «A survey on fault tolerance techniques in cloud computing,» *International Journal of Science, Engineering and Technology*, vol. 3, n.º Issue 2, pp. 1–62, 2015.

[40] A. Bhavsar e A. More, «A Holistic Approach to Autonomic Self-Healing Distributed Computing System,» vol. 76, n.º 3, pp. 25–30, 2013.

[41] G. M. Nayeem e M. J. Alam, «Analysis of Different Software Fault Tolerancen Techniques,» 2006.

[42] P. Pinto. (2016). Tutorial – Balanceamento de carga em servidores com HAProxy. accessed 02-11-2018, URL: https://pplware.sapo.pt/.

[43] HAProxy. (2018). The Reliable, High Performance TCP/HTTP Load Balancer. accessed 02-04-2018, URL: http://www.haproxy.org/#docs.

[44] A. Cassen. (2019). What is Keepalived? accessed 08-18-2018, URL: https://www.keepalived.org/.

[45] D. L. S. Martins, «Implementação de cloud privada,» tese de mestrado, Universidade de Coimbra, Faculdade de Ciências e Tecnologia, Coimbra, 2014, p. 163.

[46] Oracle. (2019). Oracle and/or its affiliates. accessed 08-18-2018, URL: https://docs.oracle.com.

[47] DEVMEDIA. (2018). MySQL: Replicação de Dados. accessed 10-11-2018, URL: https://www.devmedia.com.br.

[48] ClusterLabs. (2018). Pacemaker - ClusterLabs. accessed 10-11-2018, URL: https://wiki.clusterlabs.org/wiki/Pacemaker.

[49] L. Perkov e N. Pavkovi, «High-Availability Using Open Source Software,» *Proceedings of the 34th International Convention*, 2011.

[50] Corosync. (2019). Corosync by corosync. accessed 02-04-2019, URL: http://corosync.github.

[51] Heartbeat. (2010). Heartbeat - Linux-HA. accessed 02-04-2019, URL: http://www.linux-ha.org/wiki/Heartbeat.

[52] WEBPAGETEST. (2019). Test a website's performance. accessed 02-02-2019, URL: https://www.webpagetest.org/.

[53] L.Vieira. (2017). Time To First Byte: o que é, como testar e qual é o ideal. accessed 02-02-2019, URL: https://blog.apiki.com.

[54] LOCUST. (2019). An open source load testing tool. accessed 02-02-2019, URL: https://locust.io/.

[55] GitHub. (2019). Newsapps Beeswithmachineguns. accessed 02-02-2019, URL: https://github.com.

[56] e. a. J. Fulmer. (2015). Siege Home. accessed 02-02-2019, URL: https://www.joedog.org/siege-home/.

[57] A. S. Foundation. (1999-2019). Apache HTTP server benchmarking tool. accessed 02-02-2019, URL: https://httpd.apache.org/.

[58] GitHub. (1999-2019). Httperf e Httperf. accessed 02-02-2019, URL: https://httpd.apache.org/.

[59] A. S. Foundation. (1999-2019). Apache JMeter. accessed 02-02-2019, URL: https://jmeter.apache.org/.