



UCS

UNIVERSIDADE DE CAXIAS DO SUL  
CENTRO DE CIÊNCIAS EXATAS E DA TECNOLOGIA  
CIÊNCIA DA COMPUTAÇÃO

BRUNO EMER

Implementação de alta  
disponibilidade em uma empresa  
prestadora de serviços para Internet

André Luis Martinotto  
Orientador

Caxias do Sul  
Maio de 2016

# **Implementação de alta disponibilidade em uma empresa prestadora de serviços para Internet**

por

Bruno Emer

Projeto de Diplomação submetido ao curso de Bacharelado em Ciência da Computação do Centro de Ciências Exatas e da Tecnologia da Universidade de Caxias do Sul, como requisito obrigatório para graduação.

## **Projeto de Diplomação**

Orientador: André Luis Martinotto

Banca examinadora:

Maria de Fatima Webber do Prado Lima

CCTI/UCS

Ricardo Vargas Dorneles

CCTI/UCS

Projeto de Diplomação apresentado em  
x de x de 2016

Daniel Luís Notari  
Coordenador

# SUMÁRIO

<b>LISTA DE SIGLAS</b>	<b>3</b>
<b>LISTA DE FIGURAS</b>	<b>4</b>
<b>LISTA DE TABELAS</b>	<b>5</b>
<b>RESUMO</b>	<b>6</b>
<b>1 INTRODUÇÃO</b>	<b>7</b>
1.1 Objetivos	8
1.2 Estrutura do trabalho	8
<b>2 ALTA DISPONIBILIDADE</b>	<b>9</b>
2.1 Tolerância a falhas	9
2.2 Redundância	11
2.3 Cálculo da alta disponibilidade	12
2.4 Considerações finais	13
<b>3 VIRTUALIZAÇÃO</b>	<b>14</b>
3.1 Máquinas virtuais de aplicação	15
3.2 Máquinas virtuais de sistema	16
3.2.1 Arquiteturas de máquinas virtuais de sistema	16
3.2.2 Implementações de máquinas virtuais de sistema	18
3.2.3 Vantagens das máquinas virtuais de sistema	19
3.3 Considerações finais	20
<b>4 ESTUDO DE CASO</b>	<b>21</b>
4.1 Ambiente físico da empresa	21
4.2 Serviços e virtualização ?	22
4.3 Serviços críticos	23
<b>REFERÊNCIAS</b>	<b>26</b>

## LISTA DE SIGLAS

ECC	<i>Error correction code</i>
ISA	<i>Instruction set architecture</i>
JVM	<i>Java virtual machine</i>
KVM	<i>Kernel-based virtual machine</i>
MTBF	<i>Mean time between failures</i>
MTTR	<i>Mean time to repair</i>
PC	<i>Personal computer</i>
RAID	<i>Redundant array of independent disks</i>
SLA	<i>Service level agreement</i>
SPOF	<i>Single point of failure</i>
TI	Tecnologia da Informação
VM	<i>Virtual machine</i>
VMM	<i>Virtual machine monitor</i>

## LISTA DE FIGURAS

Figura 3.1:	Máquinas virtuais de aplicação e de sistema. . . . .	15
Figura 3.2:	Componentes da virtualização. . . . .	17
Figura 3.3:	Arquiteturas de máquinas virtuais de sistema. COLOCAR host OS ABAIXO DO HIPERVISOR ??? . . . . .	17
Figura 3.4:	Implementações de máquinas virtuais de sistema. . . . .	18
Figura 4.1:	Imagem do <i>rack</i> e dos servidores. . . . .	24
Figura 4.2:	Estrutura física. . . . .	24
Figura 4.3:	Servidor virtualização Brina. REFAZER ??? . . . . .	24
Figura 4.4:	Servidor virtualização Fulmine. . . . .	25

## LISTA DE TABELAS

Tabela 2.1: Níveis de alta disponibilidade e exemplos de sistemas . . . . .	12
Tabela 4.1: Configuração dos servidores físicos. . . . .	22

## RESUMO

Palavras-chave: .

# 1 INTRODUÇÃO

O crescente avanço tecnológico e o desenvolvimento da internet, provocou um aumento no número de aplicações ou serviços que dependem da infraestrutura de Tecnologia da Informação (TI). Além disso, percebe-se um aumento significativo no número de operações e negócios *on-line* que são realizados, tanto por organizações públicas ou privadas, quanto por grande parte da população.

Desta forma, a sociedade está cada vez mais dependente da tecnologia, de computadores e de sistemas. De fato, pode-se observar sistemas computacionais desde em uma farmácia, até em uma grande indústria. Sendo assim, a estabilidade e a disponibilidade destes sistemas apresenta um grande impacto em nosso dia-a-dia, pois um grande número de atividades cotidianas dependem deles.

Uma interrupção imprevista em um ambiente computacional poderá causar um prejuízo financeiro para a empresa que fornece o serviço, além de interferir na vida das pessoas que dependem de forma direta ou indireta deste serviço. Essa interrupção terá maior relevância para corporações cujo o serviço ou produto final é fornecido através da internet, como por exemplo, o comércio eletrônico, *web sites*, sistemas corporativos, entre outros. Em um ambiente extremo, pode-se imaginar o caos e o possível risco de perda de vidas que ocorreria em caso de uma falha em um sistema de controle aéreo (COSTA, 2009).

Para essas empresas um plano de contingência é fundamental para garantir uma boa qualidade de serviço, além de otimizar o desempenho das atividades, e também para fazer uma prevenção de falhas e uma recuperação rápida caso essas ocorram (COSTA, 2009). De fato, hoje em dia a confiança em um serviço é um grande diferencial para a empresa fornecedora deste serviço, sendo que a alta disponibilidade é fundamental para atingir este objetivo.

A alta disponibilidade consiste em manter um sistema 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* (REIS, 2009). Para que um sistema seja altamente disponível ele deve ser tolerante a falhas, sendo que a tolerância a falhas é frequentemente implementada, utilizando redundância. No caso de uma falha em um dos componentes evita-se a interrupção do sistema, uma vez que o sistema poderá continuar funcionando, utilizando o outro componente (BATISTA, 2007).

Neste trabalho será realizado um estudo sobre a implementação de um sistema de alta disponibilidade em uma empresa de hospedagens. Essa empresa oferece serviços pela internet, como por exemplo hospedagens de sites, *e-mail*, sistemas de gestão, *e-mail marketing*, entre outros. A empresa possui aproximadamente 60 servidores físicos e virtuais, e aproximadamente 9000 clientes, sendo que em períodos de pico



atende em torno de 1000 requisições por segundo.

Atualmente, a empresa possui redundância de conexões de acesso a internet, refrigeração e energia, com *nobreaks* e geradores. Porém, essa empresa não possui nenhuma redundância nos serviços que estão sendo executados nos servidores. Desta forma, caso ocorra uma falha de *software* ou de *hardware*, os serviços ficarão indisponíveis. Neste trabalho será realizada uma análise dos serviços oferecidos pela empresa, sendo que mecanismos de alta disponibilidade serão desenvolvidos para os serviços mais críticos. Para a redução dos custos serão utilizadas ferramentas gratuitas e de código aberto.

## 1.1 Objetivos

Atualmente a empresa estudada não possui nenhuma solução de alta disponibilidade para seus serviços críticos. Desta forma, neste trabalho será desenvolvida uma solução de alta disponibilidade para estes serviços, sendo que essa solução será baseada no uso de ferramentas de código aberto e de baixo custo. Para que o objetivo geral seja atendido os seguintes objetivos específicos deverão ser realizados:

- Identificar os serviços críticos a serem integrados ao ambiente de alta disponibilidade;
- Definir as ferramentas a serem utilizadas para implementar tolerância a falhas;
- Realizar testes para a validação do sistema de alta disponibilidade que foi desenvolvido.

## 1.2 Estrutura do trabalho

O trabalho foi estruturado em cinco capítulos, que estão detalhados a seguir:

- O Capítulo 2 apresenta o conceito de alta disponibilidade e outros conceitos relacionados;
- No Capítulo 3 é descrito um breve histórico da virtualização, bem como o conceito de máquinas virtuais e também as suas classificações e estratégias;
- No Capítulo 4 é apresentado o ambiente atual da empresa, seus serviços que são fornecidos e a identificação dos serviços críticos.

## 2 ALTA DISPONIBILIDADE

Alta disponibilidade é um termo conhecido, sendo cada vez mais empregada em ambientes computacionais. O objetivo de prover alta disponibilidade resume-se em garantir que um serviço esteja sempre disponível quando o cliente solicitar ou acessar (COSTA, 2009). A alta disponibilidade geralmente é implementada com uma redundância de *hardware* ou de *software* para que o serviço fique mais tempo disponível, sendo que quanto maior for a disponibilidade desejada maior deverá ser a redundância no ambiente, assim reduzindo os pontos únicos de falha, que em inglês são chamados de *Single point of failure* (SPOF). A alta disponibilidade está diretamente relacionada aos conceitos de:

- Dependabilidade: indica a qualidade do serviço fornecido e a confiança depositada neste serviço. A dependabilidade envolve atributos como segurança de funcionamento, segurança de acesso, manutenabilidade, testabilidade e comprometimento do desempenho (WEBER, 2002);
- Confiabilidade: é o atributo mais importante, pois transmite a ideia de continuidade de serviço (PANKAJ, 1994). A confiabilidade refere-se a probabilidade de um serviço estar funcionando corretamente durante um dado intervalo de tempo;
- Disponibilidade: é a probabilidade de um serviço estar operacional no instante em que for solicitado (COSTA, 2009);
- Tolerância a falhas: procura garantir a disponibilidade de um serviço utilizando mecanismos capazes de detectar, mascarar e recuperar falhas, e seu objetivo é alcançar a dependabilidade, indicando uma boa qualidade de serviço (COSTA, 2009). A tolerância a falhas é um dos principais conceitos da alta disponibilidade, sendo descrita na Seção 2.1.

### 2.1 Tolerância a falhas

Sabe-se que o *hardware* tende a falhar, principalmente devido a fatores físicos, por isso utiliza-se métodos para a prevenção de falhas. A abordagem de prevenção de falhas é realizada na etapa de projeto, ou seja, consiste em definir mecanismos que impeçam que as falhas ocorram. Além disso, a prevenção de falhas melhora a disponibilidade e a confiabilidade de um serviço, uma vez que essa tem como objetivo prever e eliminar o maior número de falhas possíveis antes de colocar o sistema em uso.

A prevenção de falhas não eliminará todas as possíveis falhas. Sendo assim, a tolerância a falhas procura fornecer a disponibilidade de um serviço mesmo com a

presença de falhas. De fato, enquanto a prevenção de falhas tem foco nas fases de projeto, teste e validação, a tolerância a falhas apresenta como foco a utilização de componentes replicados para mascarar as falhas (PANKAJ, 1994).

O objetivo da tolerância a falhas é aumentar a disponibilidade de um sistema, ou seja, aumentar o intervalo de tempo em que os serviços fornecidos estão disponíveis aos usuários. Um sistema é dito tolerante a falhas se ele for capaz de mascarar a presença de falhas ou recuperar-se de uma falha sem afetar o funcionamento do sistema. A tolerância a falhas é implementada utilizando redundância (Seção 2.2). Um exemplo muito utilizado para tornar um sistema tolerante a falhas é a virtualização. Nestes ambientes normalmente existem dois servidores físicos onde máquinas virtuais são executadas, sendo que no caso de um dos servidores falhar, o *software* de monitoramento fará a transferência das máquinas virtuais para o outro servidor, de forma transparente aos usuários, evitando assim a indisponibilidade do serviço. Os principais conceitos de virtualização, são apresentados no Capítulo 3.

A tolerância a falhas pode ser dividida em dois tipos. O primeiro tipo, o mascaramento, não se manifesta na forma de erro ao sistema, pois as falhas são tratadas na origem. O mascaramento é utilizado principalmente em sistemas de tempo real crítico. Um exemplo são os códigos de correção de erros, em inglês *Error correction code* (ECC), que são utilizados em memórias para detecção e correção de erros.

O segundo tipo consiste em detectar, localizar a falha, e reconfigurar o *software* ou *hardware* de forma a corrigir a falha. Esse tipo de tolerância a falha é dividido nas seguintes etapas (WEBER, 2002).

- Detecção: realiza o monitoramento e aguarda uma falha se manifestar em forma de erro, para então passar para a próxima fase. Um exemplo de detecção de erro é um cão de guarda (*watchdog timer*), que recebe um sinal do programa ou serviço que está sendo monitorado e caso este sinal não seja recebido, o *watchdog* irá se manifestar na forma de erro. Um outro exemplo é o esquema de duplicação e comparação, onde são realizadas operações em componentes replicados com mesmos dados de entrada, e então os dados de saída são comparados. No caso de diferenças nos dados de saída um erro é gerado.
- Confinamento: responsável pela restrição de um erro para que dados inválidos não se propaguem para todo o sistema, pois entre a falha e a detecção do erro há um intervalo de tempo. Neste intervalo pode ocorrer a propagação do erro para outros componentes do sistema, sendo assim antes de executar medidas corretivas é necessário definir os limites da propagação. Na fase de projeto essas restrições devem ser previstas e tratadas. Um exemplo de confinamento é o isolamento de alguns processos que estão em execução em um sistema operacional, sendo que esse sistema faz o gerenciamento dos processos para isolar e impedir que as falhas de um processo gerem problemas em outros processos.
- Recuperação: após a detecção de um erro ocorre a recuperação, onde o estado de erro é alterado para estado livre de erros. A recuperação pode ser feita de duas formas, que são:
  - *forward error recovery* (recuperação por avanço): ocorre uma condução para um estado que ainda não ocorreu. É a forma de recuperação mais eficiente, porém mais complexa de ser implementada.

- *backward error recovery* (recuperação por retorno): ocorre um retorno para um estado anterior livre de erros. Para retornar ao estado anterior podem ser utilizados pontos de recuperação (*checkpoints*). Assim quando ocorrer um erro, um *rollback* é executado, ou seja, o sistema retornará a um estado anterior a falha.
- Tratamento: procura prevenir que futuros erros aconteçam. Nesta fase ocorre a localização da falha para descobrir o componente que originou a falha. A substituição do componente danificado pode ser feita de forma manual ou automática. O reparo manual é feito por um operador que é responsável pelo reparo ou substituição de um componente. Como exemplo pode-se citar a troca de um disco rígido de um servidor. Já o reparo automático é utilizado quando existe um componente em espera para a substituição, como por exemplo, um disco configurado como *hot spare*, ou seja, um componente de *backup* que assumirá o lugar do outro imediatamente após o componente principal falhar. Em *storages* ou servidores, o *hot spare* pode ser configurado através de um *Redundant array of independent disks* (RAID) (ROUSE, 2013).

## 2.2 Redundância

A redundância pode ser implementada através da replicação de componentes, e apresenta como objetivo reduzir o número de SPOF e garantir o mascaramento de falhas. Na prática, se um componente falhar ele deve ser reparado ou substituído por um novo, sem que haja uma interrupção no serviço. Além disso, a redundância pode ser implementada através do envio de sinais ou *bits* de controle junto aos dados, servindo assim para detecção e correção de erros (WEBER, 2002). Segundo (NØRVÅG, 2000) existem quatro tipos diferentes de redundância que são:

- *Hardware*: utiliza-se a replicação de componentes, sendo que no caso de falha em um deles o outro possa assumir seu lugar. Para fazer a detecção de erros a saída de cada componente é constantemente monitorada e comparada à saída do outro componente. Um exemplo prático de redundância de *hardware* são os servidores com fontes redundantes. Nestes são utilizadas duas fontes ligadas em paralelo, sendo que caso uma falhe a outra suprirá a necessidade de todo o servidor;
- *Informação*: ocorre quando uma informação extra é enviada ou armazenada para possibilitar a detecção e a correção de erros. Um exemplo são os *checksums* (soma de verificação). Esses são calculados antes da transmissão ou armazenamento dos dados e recalculados ao recebê-los ou recuperá-los, assim sendo possível verificar a integridade dos dados. Outro exemplo bastante comum são os *bits* de paridade que são utilizados para detectar falhas que afetam apenas um *bit* (WEBER, 2002);
- *Software*: pode-se definir redundância de *software* como a configuração de um serviço ou *software* em dois ou mais locais diferentes. Pode-se citar como exemplo, um sistema gerenciador de banco de dados *MySQL*, que pode ser configurado com um modelo de replicação do tipo *master-slave*, onde um servidor principal (*master*) grava as operações em um arquivo, para que então os servidores *slaves*, possam recuperar e executar essas operações, com isso mantendo

Nível	Uptime	Downtime por ano	Exemplos
1	90%	36.5 dias	computadores pessoais
2	98%	7.3 dias	
3	99%	3.65 dias	sistemas de acesso
4	99.8%	17 horas e 30 minutos	
5	99.9%	8 horas e 45 minutos	provedores de acesso
6	99.99%	52.5 minutos	CPD, sistemas de negócios
7	99.999%	5.25 minutos	sistemas de telefonia ou bancários
8	99.9999%	31.5 minutos	sistemas de defesa militar

Tabela 2.1: Níveis de alta disponibilidade e exemplos de sistemas

os dados sincronizados. Neste caso, tanto o servidor *master* quanto os *slaves* executam o serviço *MySQL*, caracterizando uma redundância (SILVA VIANA, 2015). A redundância de *software* também pode ser implementada com o objetivo de tolerar falhas e *bugs* em um *software* crítico. Existem algumas técnicas que podem ser utilizadas para isso, como por exemplo, a programação de *n*-versões, que consiste no desenvolvimento de *n* versões de um mesmo *software*, desta forma, possibilita-se o aumento da disponibilidade, uma vez que elas provavelmente não apresentarão os mesmos erros. A programação de *n*-versões possui um custo muito elevado, não sendo muito utilizada.

- Tempo: este é feito através da repetição de um conjunto de instruções em um mesmo componente, assim detectando uma falha caso ocorra. Essa técnica necessita tempo adicional, e é utilizada em sistemas onde o tempo não é crítico. Por exemplo, um *software* de monitoramento de serviços em servidores que faz um teste em cada serviço. No caso de ocorrência uma falha em um serviço, uma ação corretiva será executada para reestabelecer este serviço. Essa técnica, diferentemente da redundância de *hardware*, não requer um *hardware* extra para sua implementação (COSTA, 2009).

## 2.3 Cálculo da alta disponibilidade

Um aspecto importante sobre alta disponibilidade é como medi-la. Para isso são utilizados os valores de *uptime* e *downtime*, que são respectivamente, o tempo em que os serviços estão em execução e o tempo em que não estão executando. A alta disponibilidade pode ser expressa pela quantidade de “noves”, isto é, se um serviço possui quatro noves de disponibilidade, este possui uma disponibilidade de 99,99% (PEREIRA FILHO, 2004).

A Tabela 2.1 apresenta alguns níveis de disponibilidade, e os seus percentuais de *Uptime*, os *Downtime* por ano. Já na última coluna tem-se alguns exemplos de serviços relacionados ao nível de disponibilidade. Pode-se observar que para alguns serviços, como por exemplo, sistemas bancários ou sistemas militares é necessário um alto nível de disponibilidade (PEREIRA FILHO, 2004).

A porcentagem de disponibilidade (*d*) pode ser calculada através da equação

$$d = \frac{MTBF}{(MTBF + MTTR)} \quad (2.1)$$

onde *Mean time between failures* (MTBF) corresponde ao tempo médio entre falhas, ou seja, corresponde ao tempo médio entre as paradas de um serviço. Já o *Mean*

*time to repair* (MTTR) é o tempo médio de recuperação, isto é, o tempo entre a queda e a recuperação de um serviço (GONÇALVES, 2009).

A alta disponibilidade é um dos principais fatores que fornece confiança aos clientes ou usuários de um serviço, sendo extremamente importante em empresas que fornecem serviços *on-line*. Por isso, as empresas desenvolveram o *Service level agreement* (SLA), que é um acordo de nível de serviço, que garante que o serviço fornecido atenda as expectativas dos clientes. Um SLA é um documento contendo uma descrição e uma definição das características mais importantes do serviço que será fornecido. Esse acordo apresenta também o percentual de disponibilidade do serviço, sendo que este deve ser minuciosamente definido. Por exemplo, um SLA deverá conter descrição do serviço, requerimentos, horário de funcionamento, *uptime* do serviço, *downtime* máximo do serviço, entre outros (SMITH, 2010).

## 2.4 Considerações finais

Neste Capítulo procurou-se definir e exemplificar a alta disponibilidade e os conceitos dependentes... Com o referencial teórico apresentado, será possível implementar a redundância de *software* através de máquinas virtuais replicadas em diferentes servidores. ???

### 3 VIRTUALIZAÇÃO

O conceito virtualização surgiu na década de 60, onde muitas vezes era necessário que um usuário utilizasse um ambiente individual, com suas próprias aplicações e totalmente isolado dos demais usuários. Esse foi um dos principais motivos para a criação de máquinas virtuais, que são conhecidas como *Virtual machine* (VM). As VMs apresentaram uma forte expansão com o sistema operacional *370*, que foi desenvolvido pela *IBM*, e foi um dos principais sistemas comerciais com suporte à virtualização da época. Esse sistema operacional executava em *mainframes*, que na época eram grandes servidores capazes de processar um grande volume de informações (LAUREANO; MAZIERO, 2008).

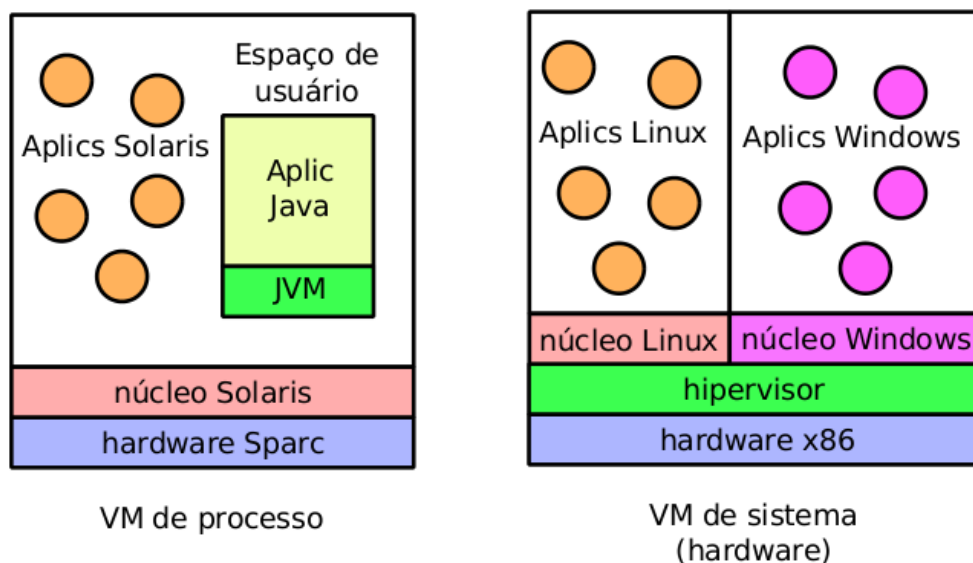
Na década de 80 houve uma redução no uso da virtualização devido a popularização do *Personal computer* (PC). Na época era mais vantajoso disponibilizar um PC para cada usuário, do que investir em *mainframes*. Devido à crescente melhora na performance do PC e ao surgimento da linguagem *Java*, no início da década de 90, a tecnologia de virtualização retornou com o conceito de virtualização de aplicação.

A virtualização foi definida nos anos 60 e 70 como uma camada entre o *hardware* e o sistema operacional que possibilitava a divisão e a proteção dos recursos físicos. Porém, atualmente ela abrange outros conceitos, como por exemplo a *Java virtual machine* (JVM), que não virtualiza um *hardware*. De fato, essa permite que uma aplicação convidada execute em diferentes tipos de sistemas operacionais.

Atualmente, define-se virtualização como uma camada de *software* que utiliza os serviços fornecidos por uma determinada interface de sistema para criar outra interface de mesmo nível. Essa camada irá permitir a comunicação entre interfaces distintas, de forma que uma aplicação desenvolvida para uma plataforma *X* possa também executar em uma plataforma *Y* (LAUREANO; MAZIERO, 2008).

??? Nesse Capítulo será apresentado as diferentes classificações de máquinas virtuais, para isso são apresentados os diferentes tipos de interfaces existentes em sistemas de computação:

- Conjunto de instruções ou *Instruction set architecture* (ISA): é a interface básica, que fica entre o *software* e o *hardware*, e é composta por instruções de código de máquina. Esta interface é dividida em dois grupos:
  - Instruções de usuário ou *User ISA*: são instruções de *hardware* disponíveis às aplicações de usuários, sendo que essas executam em modo não-privilegiado;
  - Instruções de sistema ou *System ISA*: são instruções exclusivamente acessíveis ao núcleo do sistema operacional. Essas são executadas em modo privilegiado. Pode-se citar as instruções de entrada e saída (*E/S*) como exemplo



Fonte: LAUREANO; MAZIERO (2008)

Figura 3.1: Máquinas virtuais de aplicação e de sistema.

de instruções de sistema;

- Chamadas de sistema ou *syscalls*: são operações oferecidas pelo núcleo do sistema operacional para as aplicações dos usuários. Essas operações permitem o acesso controlado aos dispositivos, memória e processador. Um exemplo de chamada de sistema é uma operação de escrita em disco.

Máquinas virtuais podem ser divididas em dois grupos principais, que são: as máquinas virtuais de aplicação (Seção 3.1), e máquinas virtuais de sistema (Seção 3.2). As máquinas virtuais de aplicação fazem a virtualização de uma aplicação e suportam apenas um processo ou aplicação, ou seja, elas provêm um ambiente onde um sistema operacional permita a execução de uma aplicação convidada. Um exemplo de máquina virtual de aplicação é a JVM. Enquanto uma máquina virtual de sistema suporta um sistema operacional convidado, com suas aplicações executando sobre ele (Figura 3.1). Uma máquina virtual executando sobre o hipervisor *Kernel-based virtual machine* (KVM) é um exemplo de máquina virtual de aplicação (LAUREANO; MAZIERO, 2008).

### 3.1 Máquinas virtuais de aplicação

As máquinas virtuais de aplicação, também chamadas de máquinas virtuais de processos, são responsáveis por prover um ambiente que permite que um sistema operacional suporte uma aplicação convidada, sendo que esta aplicação possui um conjunto de instruções, ou de chamadas do sistema, diferentes da arquitetura do sistema hospedeiro. Neste caso, quando temos chamadas do sistema operacional ou instruções de máquina que são diferentes das oferecidas pela máquina real, será necessário uma tradução dessas interfaces, que será feita pela camada de virtualização. Existem dois tipos de máquinas virtuais de aplicação, que são:



- Máquinas virtuais de linguagem de alto nível: esse tipo de máquina virtual foi criado levando em consideração uma linguagem de programação e o seu compilador. Neste caso, o código compilado gera um código binário que não pode ser executado em uma arquitetura real, mas pode ser executado em uma máquina virtual. Sendo assim para cada arquitetura ou sistema operacional deverá existir uma máquina virtual que permita a execução da aplicação nesse ambiente. Como exemplo deste tipo de máquina virtual pode-se citar a máquina virtual Java (JVM) e a *Microsoft Common Language Infrastructure*, que é a base da plataforma *.NET* (CARISSIMI, 2008);
- Emulação no sistema operacional: nesse caso é feito um mapeamento entre as chamadas de sistema que são utilizadas pela aplicação e as chamadas do sistema operacional real. A virtualização de aplicação pode ser encontrada em ferramentas que emulam uma aplicação desenvolvida para uma plataforma em outra plataforma. Como exemplo, pode-se citar o *Wine* (WineHQ, 2016), que permite executar aplicações *Windows* em plataformas *Linux*.

## 3.2 Máquinas virtuais de sistema

As máquinas virtuais de sistema, também chamadas de hipervisor ou *Virtual machine monitor* (VMM), são uma camada de *software* que possibilita que um ou mais sistemas operacionais convidados executem independentemente sobre um mesmo computador físico. O hipervisor provê uma interface ISA virtual, que pode ou não ser igual a interface real, e virtualiza outros componentes de *hardware*, para que cada máquina virtual convidada possa ter seus próprios recursos.

A virtualização de sistema utiliza abstrações em sua arquitetura. Por exemplo, ela transforma um disco rígido físico em dois discos rígidos virtuais menores, sendo que esses discos virtuais são arquivos armazenados no disco físico. Sabendo que arquivos são uma abstração em um disco rígido físico, pode-se dizer que virtualização não é apenas uma camada de abstração do *hardware*, ela faz a reprodução do *hardware* (SMITH; NAIR, 2005).

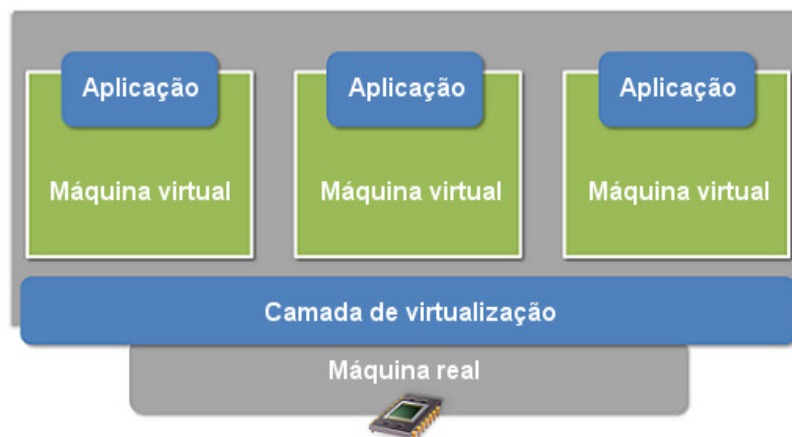
Nesse modelo, o ambiente de virtualização de sistema é composto basicamente por três componentes (Figura 3.2):

- Máquina real: também chamado de hospedeiro, que é o *hardware* onde o sistema de virtualização irá executar;
- Camada de virtualização: é conhecida como hipervisor ou também chamada de VMM. Essa camada tem como função criar interfaces virtuais, para a comunicação da máquina virtual com a máquina real;
- Máquina virtual: também conhecido como sistema convidado, que executa sobre a camada de virtualização. Sendo que, geralmente existem várias máquinas virtuais executando simultaneamente.

### 3.2.1 Arquiteturas de máquinas virtuais de sistema

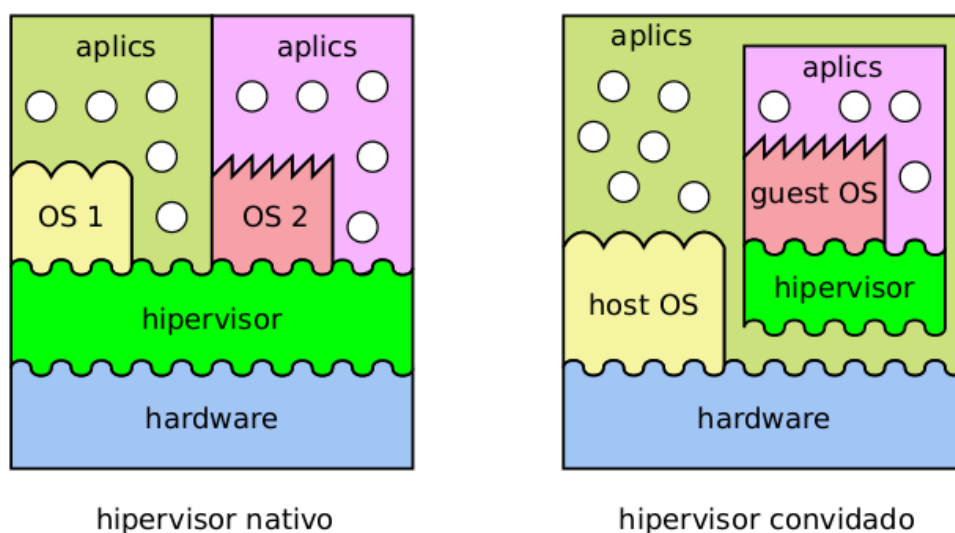
Existem basicamente duas arquiteturas de hipervisor de sistema, que são apresentadas na Figura 3.3 (MAZIERO, 2013):

- Hipervisores nativos: esse hipervisor executa diretamente sobre o *hardware*, ou



Fonte: ANDRADE (2011)

Figura 3.2: Componentes da virtualização.



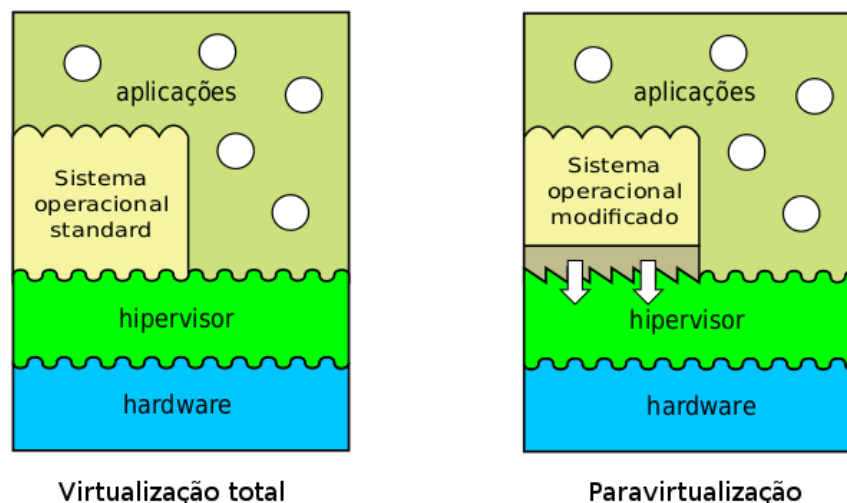
Fonte: MAZIERO (2013)

Figura 3.3: Arquiteturas de máquinas virtuais de sistema. COLOCAR host OS ABAIXO DO HIPERVISOR ???

seja, sem um sistema operacional hospedeiro. Neste caso, o hipervisor nativo faz a multiplexação dos recursos do *hardware* (memória, disco rígido, interface de rede, entre outros) e disponibiliza esses recursos para as máquinas virtuais. Alguns exemplos que utilizam esse hipervisor são *IBM 370* (IBM, 2016), o *Xen* (Citrix, 2016a) e o *VMware ESXi* (VMware, 2016a);

- Hipervisores convidados: esse tipo de hipervisor executa sobre um sistema operacional hospedeiro, e utiliza os recursos desse sistema para gerar recursos virtuais para as máquinas virtuais. Normalmente esse tipo suporta apenas um sistema operacional convidado para cada hipervisor. Exemplos de *softwares* que possuem esse tipo de arquitetura são: o *VirtualBox* (Oracle, 2016), o *VMware Player* (VMware, 2016b) e o *QEmu* (QEmu, 2016).

Destaca-se que hipervisores convidados são mais flexíveis que os nativos, pois podem ser implementados em diversos sistemas operacionais e *hardwares*. Já os



Fonte: MAZIERO (2013)

Figura 3.4: Implementações de máquinas virtuais de sistema.

hipervisores nativos possuem melhor desempenho pois acessam o *hardware* diretamente.

### 3.2.2 Implementações de máquinas virtuais de sistema

As máquinas virtuais de sistema podem ser implementadas usando diferentes tipos de estratégias. Atualmente as estratégias mais utilizadas são a virtualização total e a paravirtualização (Figura 3.4), detalhadas a seguir:

- **Virtualização total:** nesta estratégia todas as interfaces de acesso ao *hardware* são virtualizadas. Desta forma, possibilita-se que sistemas operacionais convidados executem como se estivessem diretamente sobre o *hardware*. Na virtualização total o conjunto de instruções do processador é acessível somente pelo hipervisor. Sendo que essa estratégia utiliza tradução dinâmica<sup>1</sup> para traduzir as instruções do sistema convidado. A grande vantagem dessa estratégia é a possibilidade de um sistema convidado ser executado sem a necessidade de ser modificado. Porém, essa estratégia possui um desempenho inferior devido ao fato do hipervisor intermediar todas as chamadas de sistemas e operações do sistema convidado. Um exemplo ferramenta que utiliza virtualização total é o *QEmu*;
- **Paravirtualização:** esta utiliza arquitetura de hipervisor nativo, sendo que essa estratégia provê um melhor acoplamento entre os sistemas operacionais convidados e o hipervisor. Para isso, o sistema convidado deve ser adaptado para o hipervisor no qual executará, ou seja, a interface de sistema (*system ISA*) será acessada diretamente pelo sistema convidado, resultando em um desempenho melhor. Um exemplo de sistema que implementa a paravirtualização é o *Xen*.

<sup>1</sup>A tradução dinâmica analisa e reorganiza as instruções de um sistema convidado para melhorar o desempenho da execução, além disso a tradução dinâmica adapta as instruções do sistema convidado para o sistema real.

A paravirtualização possui um desempenho superior se comparada a virtualização total, pois acessa alguns recursos de forma direta, sendo que o hipervisor é reponsável somente por impedir que o sistema convidado execute operações indevidas. Pode-se citar como exemplo o controle de acesso à memória feito pelo hipervisor. Na virtualização total o hipervisor reserva um espaço para cada sistema convidado, que por sua vez acessa a memória como se fosse uma memória física, que inicia o seu endereçamento na posição zero. Sendo assim, cada vez que o sistema convidado acessar a memória, o hipervisor precisará converter os endereços do sistema convidado para os endereços reais de memória. Na paravirtualização o hipervisor informa ao sistema convidado a área de memória que ele pode utilizar, assim não sendo necessário nenhuma conversão de endereços.

Apesar de apresentar um desempenho inferior, a virtualização total possui uma maior portabilidade, ou seja, permite que sistemas operacionais executem como convidados, sem a necessidade de serem modificados. Pode-se, por exemplo, transferir um sistema operacional instalado diretamente em uma máquina física para um ambiente virtual, sem a necessidade de reinstalá-lo e reconfigurar todas as aplicações.

### 3.2.3 Vantagens das máquinas virtuais de sistema

A portabilidade é uma das grandes vantagens da virtualização, que também pode ser aplicada em *desktops*. Pode-se citar como exemplo o desenvolvimento de *software* para diversos sistemas operacionais sem a necessidade de um computador para cada sistema operacional. Assim, máquinas virtuais em *desktops* podem ser utilizadas em ambientes de desenvolvimento, pois possibilitam a execução de múltiplas plataformas sem comprometer o sistema operacional original (CARISSIMI, 2008). Pode-se citar como exemplo o *VMware Workstation*, que possibilita a virtualização em PC para fins de desenvolvimento de *software* (WMware, 2016).

Em empresas pode-se utilizar virtualização de *desktops*, através da configuração de terminais remotos nos computadores e um servidor para centralizar as máquinas virtuais. Com isso torna-se mais simples a manutenção dos *desktops* e exige um *hardware* de menor valor, além disso essa técnica possibilita uma maior segurança dos dados. Exemplos desse tipo de virtualização são o *Xen Desktop* (Citrix, 2016b) e o *VMware Horizon View* (VMware, 2016c).

Pode-se também encontrar virtualização de *desktops* em laboratórios de universidades, devido a necessidade de executar diferentes sistemas operacionais para determinadas disciplinas. Isso é necessário quando pretende-se configurar e executar aplicações para fim de experimentos ou aprendizagem, com isso essas ações não afetarão o sistema hospedeiro, pois estarão executando no sistema operacional de uma máquina virtual. O benefício desse tipo de cenário é a facilidade de manipulação das máquinas virtuais, pois podem ser restauradas de uma forma simples.

Em muitos casos as empresas utilizam serviços distribuídos entre diferentes servidores físicos, como, por exemplo, servidores de e-mail, hospedagens e banco de dados, com isso existe uma grande ociosidade de recursos. Portanto, uma das grandes vantagens da virtualização é um melhor aproveitamento dos recursos. De fato, alocando vários serviços em um único servidor físico e tem-se um melhor aproveitamento do *hardware* (MOREIRA, 2006). Além disso, pode-se ter uma redução de custos com a administração e a manutenção dos servidores físicos. Em um ambiente heterogêneo pode-se também utilizar virtualização, pois ela permite a instalação de diversos sistemas operacionais em um único servidor. Esse tipo de virtualização fa-

vorece a implementação do conceito “um servidor por serviço”, que consiste em ter um servidor para cada serviço. Além disso tem-se o isolamento de serviços, ou seja, caso ocorra uma falha de segurança em um serviço, essa falha não comprometerá todo o sistema (CARISSIMI, 2008).

Outra motivação para a utilização de virtualização consiste no custo da energia elétrica. A economia de energia pode ser obtida através da implantação de servidores mais robustos para substituir dezenas de servidores comuns. Outros fatores como refrigeração do ambiente e espaço físico utilizado também podem ser reduzidos com a implantação de virtualização de servidores, e conseqüentemente, reduzem os custos de energia.

### **3.3 Considerações finais**

Neste Capítulo...

## 4 ESTUDO DE CASO

Neste trabalho será feito um estudo sobre uma empresa que fornece serviços de hospedagens e também está associada a um provedor de internet<sup>1</sup>. A empresa possui grande parte de seus clientes localizados na serra do Rio Grande do Sul, sendo que o número de clientes é aproximadamente 9000. A sede da empresa está localizada na cidade de Garibaldi, além disso possui quatro filiais no estado, atendendo aproximadamente xx cidades. Essa empresa, que será o foco deste trabalho, possui aproximadamente xx funcionários.

Essa empresa oferece serviços pela internet aos seus clientes, que são: hospedagens de sites, banco de dados, *e-mail*, sistemas de gestão, *e-mail marketing*, *backup*, *máquinas virtuais*, autenticação *ADSL*, rádio *online* e telefonia. O provedor associado fornece aos seus clientes acesso à internet via rádio e acesso à internet por meio de fibra óptica.

Sabendo que a empresa atualmente possui redundância de refrigeração e energia. A redundância de refrigeração é composta por três ares-condicionados. A redundância de energia é feita através de três *nobreaks*, sendo que dois deles fazem o balanceamento de carga dos servidores e outros equipamentos como por exemplo roteadores. Além disso, possui dois geradores para suprir a necessidade de consumo de energia elétrica. Assim, essas redundâncias tem como objetivo prover um ambiente físico estável e confiável.

Para ser possível propor uma solução de alta disponibilidade nos servidores é necessário conhecer o cenário atual da empresa, além de selecionar os serviços mais críticos para a empresa. Nas próximas seções serão detalhados: o ambiente físico dos servidores, com suas estruturas e suas configurações (Seção 4.1); a estrutura de virtualização, com a relação de servidores físicos e virtuais, e todos os serviços fornecidos pela empresa (Seção 4.2); e por fim a seleção dos serviços críticos (Seção 4.3).

### 4.1 Ambiente físico da empresa

A estrutura atual da empresa é composta por quatorze servidores físicos montados em um *rack* (Figura 4.1). A configuração de *hardware* desses servidores esta listado na Tabela 4.1, que possui o nome do servidor, quantidade e modelo de processadores, quantidade e tipo de memória, número de discos e a capacidade unitária, e o modelo do servidor.

---

<sup>1</sup>É importante salientar que esse provedor utiliza a maior parte dos serviços da empresa, pois possui maior número de clientes.

Servidor	Processador	Memória	Disco	Modelo
bello	1 x Intel Core 2 Duo CPU E6750 2.66 GHz	2 GB DDR2	5,5 TB SATA	
brina	2 x Intel Xeon CPU E5410 2.33 GHz	24 GB DDR2	6 x 300 GB SAS	Dell PowerEdge 2950
cacti	2 x Intel Xeon CPU E5310 1.60 GHz	12 GB DDR2	2 x 73 GB SAS	Dell PowerEdge 2950
dati	2 x Intel Xeon CPU 3.20 GHz	4 GB DDR2	2 x 146 GB SCSI	Dell PowerEdge 1850
fulmine	1 x Intel Xeon CPU E5-2650 2.00 GHz	32 GB DDR3	6 x 2 TB SATA	IBM System x3650 M4
monit	1 x Intel Core 2 Quad CPU Q9550 2.83 GHz	4 GB DDR2	120 GB SSD	
nino	1 x Intel Core 2 Duo CPU E4500 2.20 GHz	4 GB DDR2	500 GB SATA	
piova	2 x Intel Xeon CPU E5530 2.40 GHz	32 GB DDR3	4 x 500G SATA	Dell PowerEdge R410
raggio	2 x Intel Xeon CPU E5630 2.53 GHz	32 GB DDR3	4 x 300 GB SAS	HP ProLiant DL360 G7
sfrunhon	1 x Intel Xeon CPU X3330 2.66 GHz	8 GB DDR2	750 GB SATA	
tempesta	2 x Intel Xeon CPU E5-2620 2.00 GHz	32 GB DDR3	5 x 1 TB SATA	Dell PowerEdge R620
tuono	2 x Intel Xeon CPU E5649 2.53 GHz	32 GB DDR3	6 x 300 GB SAS 2 x 146 GB SAS	HP ProLiant DL380 G7
venti	1 x Intel Xeon CPU E3-1220 3.10 GHz	16 GB DDR3	2 x 3 TB SATA	Dell PowerEdge R210 II
vigilante	1 x Intel Pentium Dual CPU E2180 2.00 GHz	4 GB DDR2	2,5 TB SATA	

Tabela 4.1: Configuração dos servidores físicos.

Os servidores utilizados para virtualização possuem redundância de *hardware*, com fonte de alimentação e discos configurados através de um RAID. Além disso, para cada servidor de virtualização dois cabos de rede são ligados a um *switch gigabit*, para possibilitar a configuração de *link aggregation*, assim caracterizando a redundância do cabeamento e dobrando a capacidade de tráfego de dados. O diagrama da Figura 4.2 demonstra uma visão geral da estrutura física dos servidores da empresa. Todos os servidores estão ligados ao *switch*, que provê aos servidores acesso à internet através do roteador. Os servidores de virtualização possuem suas respectivas VMs, que executam aplicações, e existem servidores que executam aplicações diretamente.

## 4.2 Serviços e virtualização ?

A empresa fornece serviços diversos, desde hospedagens de sites até *DNS* recrusivo para um provedor de internet. Atualmente sete servidores são utilizados com a virtualização, e os outros sete possuem aplicações executando diretamente. Sendo que existem quarenta e seis VMs distribuídas entre os sete servidores de virtualização, esses servidores são:

- Servidor Brina: possui quatro VMs, que executam os serviços de ... (Figura 4.3);
- Servidor Fulmine: este executa doze VMs, que executam: ... (Figura 4.4);
- Servidor Piova:
- Servidor Raggio:
- Servidor Tempesta:

- Servidor Tuono:
- Servidor Venti:

Todos os servidores de virtualização possuem o sistema operacional *Ubuntu* versão *14.04 LTS*, para virtualização utiliza-se o hipervisor KVM e o *QEmu*, ambos são projetos de *software* livre, para compor o ambiente de virtualização...

-graficos cpu memoria disco cada servidor de virtualizacao? colocar aqui ou na implementacao?

A maioria dos serviços são fornecidos por meio de *software* de código aberto, e a maioria dos sistemas operacionais são *Ubuntu* ou outros sistemas de código aberto, que também são *softwares* livre.

### 4.3 Serviços críticos

Esboço:

-DNS (impacto direto para clientes e rede interna):

requisicoes por segundo

numero de usuarios

-Radius (impacto direto para clientes):

numero de usuarios autenticados em x tempo

quantidade de dados armazenados no db em x tempo, trafego utilizado, tempo conexao

numero de usuarios

-Sistemas (impacto indireto para clientes):

gasto com funcionarios ociosos

quantidade de atendimento a clientes

numero de cobranças enviadas para clientes efetuar pagamento

comunicacao entre setores e funcionarios

numero de usuarios

-Telefonia (impacto indireto para clientes):

quantidade de atendimento a clientes

comunicacao entre setores e funcionarios

ligacoes saintes, atendimento, cobranca, tecnicos instalacoes internet

numero de usuarios

Exibir disponibilidade média atual dos serviços, com gráficos



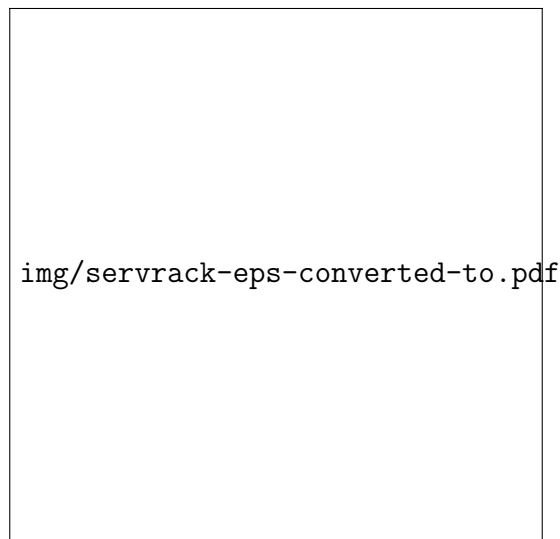


Figura 4.1: Imagem do *rack* e dos servidores.

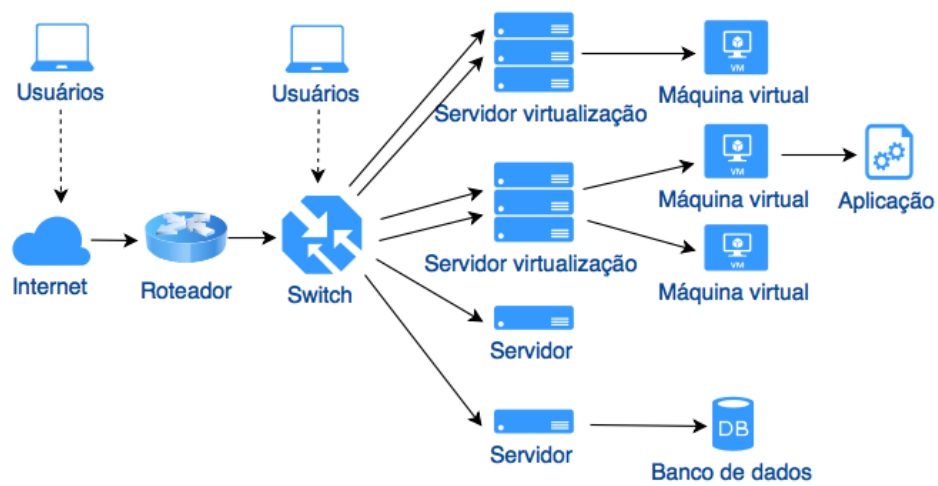


Figura 4.2: Estrutura física.

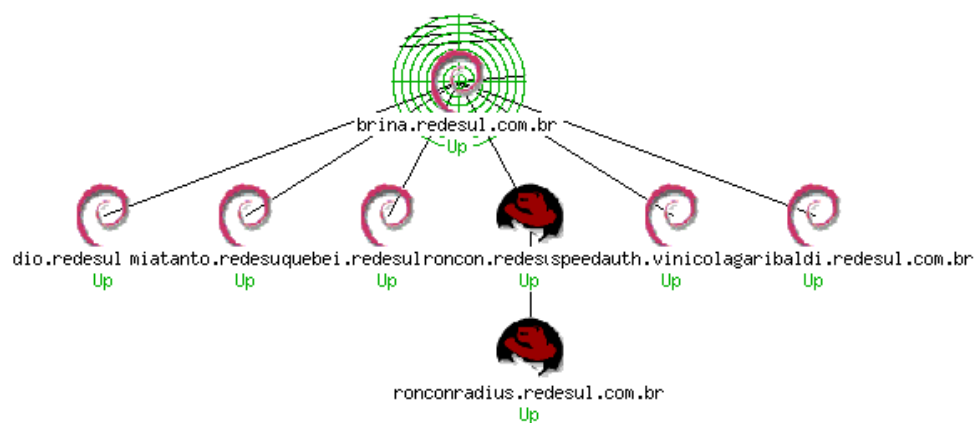


Figura 4.3: Servidor virtualização Brina. REFAZER ???



Figura 4.4: Servidor virtualização Fulmine.

## REFERÊNCIAS

ANDRADE, L. **Visão geral sobre virtualização**. <Disponível em: <https://tecnologiasemsegredos.wordpress.com/category/virtualizacao/>>. Acesso em 21 de maio de 2016.

BATISTA, A. C. **Estudo teórico sobre cluster linux**. 2007. Pós-Graduação(Administração em Redes Linux) — Universidade Federal de Lavras, Minas Gerais.

CARISSIMI, A. Virtualização: da teoria a soluções. In: **Minicursos do Simpósio Brasileiro de Redes de Computadores**. Porto Alegre: XXVI Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos, 2008.

Citrix. **The Xen Project, the powerful open source industry standard for virtualization**. <Disponível em: <http://www.xenproject.org/>>. Acesso em 22 de maio de 2016.

Citrix. **XenApp and XenDesktop - Virtual Apps and Desktops**. <Disponível em: <https://www.citrix.com/products/xenapp-xendesktop/>>. Acesso em 22 de maio de 2016.

COSTA, H. L. A. **Alta disponibilidade e balanceamento de carga para melhoria de sistemas computacionais críticos usando software livre: um estudo de caso**. 2009. Pós-Graduação em Ciência da Computação — Universidade Federal de Viçosa, Minas Gerais.

GONÇALVES, E. M. **Implementação de Alta disponibilidade em máquinas virtuais utilizando Software Livre**. 2009. Trabalho de Conclusão (Curso de Engenharia da Computação) — Faculdade de Tecnologia e Ciências Sociais Aplicadas, Brasília.

IBM. **System/370 Model 145**. <Disponível em: [https://www-03.ibm.com/ibm/history/exhibits/mainframe/mainframe\\_PP3145.html](https://www-03.ibm.com/ibm/history/exhibits/mainframe/mainframe_PP3145.html)>. Acesso em 22 de maio de 2016.

LAUREANO, M. A. P.; MAZIERO, C. A. Virtualização: conceitos e aplicações em segurança. In: **Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais**. Gramado - Rio Grande do Sul: VIII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais, 2008.

MAZIERO, C. A. **Sistemas Operacionais: conceitos e mecanismos**. 2013. Dissertação (Mestrado em Ciência da Computação) — DAIInf UTFPR, Paraná.

MOREIRA, D. **Virtualização: rode vários sistemas operacionais na mesma máquina**. <Disponível em: <http://idgnow.com.br/ti-corporativa/2006/08/01/idgnoticia.2006-07-31.7918579158/#&panel1-3>>. Acesso em 5 de abril de 2016.

NØRVÂG, K. **An Introduction to Fault-Tolerant Systems**. 2000. IDI Technical Report 6/99 — Norwegian University of Science and Technology, Trondheim, Noruega.

Oracle. **Oracle VM VirtualBox**. <Disponível em: <https://www.virtualbox.org/>>. Acesso em 22 de maio de 2016.

PANKAJ, J. **Fault tolerance in distributed system**. Nova Jérsei, Estados Unidos: P T R Prentice Hall, 1994.

PEREIRA FILHO, N. A. **Serviço de pertinência para clusters de alta disponibilidade**. 2004. Dissertação para Mestrado em Ciência da Computação — Universidade de São Paulo, São Paulo.

QEmu. **QEMU open source processor emulator**. <Disponível em: [http://wiki.qemu.org/Main\\_Page](http://wiki.qemu.org/Main_Page)>. Acesso em 22 de maio de 2016.

REIS, W. S. dos. **Virtualização de serviços baseado em contêineres: uma proposta para alta disponibilidade de serviços em redes linux de pequeno porte**. 2009. Monografia Pós-Graduação(Administração em Redes Linux) — Apresentada ao Departamento de Ciência da Computação, Minas Gerais.

ROUSE, M. **Hot spare**. <Disponível em: <http://searchstorage.techtarget.com/definition/hot-spare>>. Acesso em 12 de abril de 2016.

SILVA VIANA, A. L. da. **MySQL: replicação de dados**. <Disponível em: <http://www.devmedia.com.br/mysql-replicacao-de-dados/22923>>. Acesso em 21 de abril de 2016.

SMITH, J. E.; NAIR, R. The architecture of virtual machines. **IEEE Computer**, [S.l.], v.38, p.32–38, 2005.

SMITH, R. **Gerenciamento de Nível de Serviço**. <Disponível em: <http://blogs.technet.com/b/ronaldosjr/archive/2010/05/25/gerenciamento-de-n-237-vel-de-servi-231-o.aspx>>. Acesso em 25 de março de 2016.

VMware. **VMware ESXi**. <Disponível em: <http://www.vmware.com/products/esxi-and-esx/overview.html>>. Acesso em 22 de maio de 2016.

VMware. **VMware Workstation Player (formerly known as Player Pro)**. <Disponível em: <https://www.vmware.com/products/player>>. Acesso em 22 de maio de 2016.

VMware. **VDI Virtual Desktop Infrastructure with Horizon**. <Disponível em: <https://www.vmware.com/products/horizon-view>>. Acesso em 22 de maio de 2016.

WEBER, T. S. **Um roteiro para exploração dos conceitos básicos de tolerância a falhas**. 2002. Curso de Especialização em Redes e Sistemas Distribuídos — UFRGS, Rio Grande do Sul.

WineHQ. **WineHQ - Execute aplicativos Windows no Linux, BSD, Solaris e Mac OS X**. <Disponível em: <https://www.winehq.org/>>. Acesso em 22 de maio de 2016.

VMware. **VMware Workstation Pro**. <Disponível em: <http://www.vmware.com/br/products/workstation>>. Acesso em 17 de maio de 2016.