UFF - UNIVERSIDADE FEDERAL FLUMINENSE

HUGO ARRAES HENLEY
SAULO REIS ZIMBARO

PROPOSTA DE RECUPERAÇÃO A DESASTRES EM SISTEMAS CRÍTICOS

NITERÓI

2014

UFF - UNIVERSIDADE FEDERAL FLUMINENSE

HUGO ARRAES HENLEY
SAULO REIS ZIMBARO

PROPOSTA DE RECUPERAÇÃO A DESASTRES EM SISTEMAS CRÍTICOS

Trabalho de Conclusão de Curso apresentado à Universidade Federal Fluminense como requisito parcial para a obtenção do grau de Bacharel em Engenharia de Telecomunicações.

Orientador:
João Marcos Meirelles da Silva

NITERÓI

2014

HUGO ARRAES HENLEY
SAULO REIS ZIMBARO

PROPOSTA DE RECUPERAÇÃO A DESASTRES EM SISTEMAS CRÍTICOS

Trabalho de Conclusão de Curso apresentado à Universidade Federal Fluminense como requisito parcial para a obtenção do grau de Bacharel em Engenharia de Telecomunicações.

Aprovada em XXXXXX de 2014.

BANCA EXAMINADORA

Prof. Dr. João Marcos Meirelles da Silva - Orientador

Prof. Dr. Carlos Alberto Malcher Bastos

Prof. Dra. Natália Castro Fernandes

Niterói
2014

Este trabalho é dedicado a toda minha família, em especial aos meus pais e irmã pelo suporte em todos os momentos, aos meus amigos pelo convívio durante toda essa jornada e a minha noiva Camila por estar sempre ao meu lado me apoiando.

Saulo Reis Zimbaro

Dedico esse trabalho a toda minha família, mas em especial aos meus pais e ao meu avô por me mostarem desde cedo a importância do estudo, sempre me apoiarem nos momentos de dificuldade e serem um grande exemplo para mim. Aos amigos, pois sem vocês essa jornada teria sido muito mais difícil. Agradeço imensamente a minha noiva Priscila por todo apoio nesses últimos anos, é maravilhoso poder contar com você.

Hugo Arraes Henley

Agradecimentos

Agradecemos ao professor João Marcos Meirelles da Silva por toda orientação que nos deu em nossa formação como Engenheiros de Telecomunicações.

Hugo Arraes Henley e Saulo Zimbaro

Resumo

Com o aumento da automação realizada através de softwares nas áreas de TI e Telecomunicações se torna cada vez mais importante o conceito de alta disponibilidade. Muitas empresas de telecomunicações utilizam softwares para controle e monitoramento de antenas, redes e equipamentos. A indisponibilidade por um grande período de tempo de algum destes pode levar a grandes prejuízos financeiros, podendo levar inclusive a falência de uma empresa.

Tendo em vista esse cenário, foi proposta uma solução em software que lida de forma simples e eficiente com a questão do *Disaster Recovery* e é aplicável a grande parte dos usuários.

Este software foi integrado à ferramenta de monitoramento NewRelic e à *Cloud* Pública da Amazon AWS através de APIs para que seja possível realizar *Disaster Recovery*.

Palavras-chave: Disaster Recovery, Cloud Computing, DR, Telecomunicações, NewRelic, Amazon AWS, IaaS.

Abstract

With increasing automation performed by software in IT and telecommunications areas, the concept of high availability become increasingly important. Many telecommunications companies use systems for monitoring and control of antennas, networks and many other equipments. The unavailability for an extended period of time can lead to major financial losses and can even lead to bankruptcy of a company.

Given this scenario, a solution was proposed using a software that deals simply and efficiently with Disaster Recovery of Web Applications and can be applicable to most users.

This software has been integrated with NewRelic and Amazon AWS via APIs, so it is possible to perform monitoring and deploy of applications as a Disaster Recovery alternative.

Keywords: Disaster Recovery, Cloud Computing, Telecommunication, Amazon AWS, IaaS, NewRelic.

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

ANATEL	:	Agência Nacional de Telecomunicações
ANSI	:	American National Standards Institute
BC	:	Business Continuity
BIA	:	Business Impact Analysis
CSDR	:	Cold site disaster recovery
DR	:	Disaster Recovery;
DNS	:	Domain Name Server
DRI	:	Disaster Recovery Institute
DaaS	:	Desktop as a service
HLR	:	Home Location Register;
HSDR	:	Hot site disaster recovery
IaaS	:	Infraestructure as a service
NMS	:	Network Management System;
NRO	:	Network Recovery Time
PaaS	:	Platform as a service
RTO	:	Recovery Time Objective;
RPO	:	Recovery Point Objective;
SaaS	:	Softare as a service
TRTO	:	Technical Recovery Time Objective
TI	:	Tecnologia da Informação
TIA	:	Telecommunications Industry Association
SLA	:	Service-level Agreement
CPU	:	Central Processing Unit
RAM	:	Random Access Memory
HD	:	Hard Disk
SSD	:	Solid-state Drive
HTTP	:	Hypertext Transfer Protocol
REST	:	Representational State Transfer
SOAP	:	Simple Object Access Protocol
API	:	Application Programming Interface
TI	:	Tecnologia da Informação

Sumário

Capítulo 1

Introdução

A sociedade está cada vez mais dependente dos sistemas de telecomunicações e isto significa que curtos períodos de inatividade deste sistemas pode resultar em significativa perda financeira ou, em alguns casos, até mesmo colocar vidas em risco. Atualmente quase toda organização confia a um sistema de telecomunicação o planejamento, gerenciamento ou ao menos o monitoramento de seus processos, recursos ou vendas. Algumas até criam sistemas e os vendem como serviços para terceiros. Dependendo da importância deste sistema se faz necessário que seja confiável e a prova de qualquer tipo de falha para que alcance seus objetivos, em outras palavras é preciso ter alta disponibilidade. O projeto e dimensionamento de um sistema com alta disponibilidade pode se tornar muito caro e de difícil monitoração, por outro lado ele se faz necessário para o sucesso do plano de negócios. A solução para este problema ainda encontra-se em aberto.

O custo do projeto de sistemas com alta disponibilidade pode ser reduzido consideravelmente quando se usa um conceito muito importante em telecomunicações, a escalabilidade. Grandes sistemas com alta complexidade e custo podem ser diminuídos, mantendo o mesma desempenho, quando se usa elementos escaláveis, ou seja, crescente e descrescente seguindo o padrão da demanda. Isto garante o uso adequado e otimizado do sistema e evita desperdícios e ociosidade de seus elementos.

Este trabalho propõe uma solução de baixo custo para controle da escalabilidade de sistemas baseados em *Cloud Computing* (Computação de Nuvem ou Computação em Nuvem ou Computação nas Nuvens , em português) . Será proposto uma solução que irá monitorar o desempenho de determinado sistema e permitirá sua expansão ou retração garantindo um nível de serviço adequado.

O presente capítulo trata dos fatores motivacionais, da importância e do objetivo

deste trabalho.

O Capítulo 2 explicará o que é *cloud computing* ou *cloud computing* e sua importância nos modelos de sistemas à prova de falhas. Exibirá as suas diferentes topologias e melhor aplicação em diferentes tipos de sistemas. Neste capítulo será comparado a adoção de um *data center* físico próprio com o uso do *cloud computing*.

O Capítulo 3 introduzirá os conceitos de alta disponibilidade e *downtime* e suas importâncias para a qualidade do serviço. Será apresentado os impactos financeiros decorrentes do *downtime* do ponto de vista de alguns setores econômicos.

O Captítulo 4 expõe o conceito de *Disaster Recovery* (DS) bem como as suas arquiteturas possíveis e impactos financeiros positivos com a adoção de políticas.

O Capítulo 5 explica a importância do monitoramento dos sistemas assim como a ferramenta NewRelic utilizada neste trabalhho.

O Captítulo 6 exibirá a nossa proposta de solução e sua aplicação. Serão discutidas as tecnologias adotadas e detalhamento da arquitetura utilizada.

O Captítulo 7 apresenta a conclusão do trabalho, contendo também os resultados obtidos com o mesmo.

1.1 Motivação

Toda empresa está sujeita a desastres, mas a maioria não está preparada para lidar com os mesmos. Desastres podem ocorrer devido a falhas humanas ou até mesmo falhas naturais, e podem impactar diretamente na reputação de uma empresa. É um grande risco manter e confiar os dados a empresas que não possuem um sistema de *Disaster Recovery* (DR) e que, por este motivo, venham a ficar indisponíveis por horas ou dias. Isto certamente impacta na confiança do mundo nesta empresa, o que está diretamente ligada ao capital que ela consegue arrecadar devido ao número de usuários que usam o seu serviço. O grande desafio é, principalmente para empresas que mantém serviços críticos para seus clientes, manter uma alta disponibilidade. É para isso que existem planos pré-definidos em caso de desastres. Um modelo muito comum é o de replicação de infra-estrutura em outro ambiente exatamente igual ao seu DataCenter principal, porém localizado a centenas de quilômetros de distância do mesmo. Esse modelo funciona, mas representa um alto custo que grande parte das empresas não é capaz de arcar, principalmente quando estão iniciando suas atividades.

Com base nesse cenário foi elaborada uma proposta de solução através de uma ferramenta que pudesse replicar o ambiente de forma automática e de baixo custo, onde a empresa só têm o custo de manter a infra-estrutura de recuperação enquanto for necessário. Esse modelo é comumente chamado de *pay-as you-go* e é adotado pela maioria das empresas que vendem infra-estrutura como serviço (IaaS).

A solução de *Disaster Recovery* aqui proposta é capaz de monitorar e restaurar uma aplicação web com facilidade em um ambiente escável com maior tolerância a falhas, como o de Cloud Computing. Para que isso seja possível, a ferramenta desenvolvida foi integrada a soluções de dois grandes players de mercado no que diz respeito a Monitoramento e *Cloud Computing*, sendo eles o *NewRelic* e a Amazon *Web Services*, respectivamente.

É esperado que com o uso da ferramenta desenvolvida o custo de manutenção de um ambiente para recuperação de desastres seja drásticamente reduzido e que a reputação da empresa devido a sua disponibilidade oferecida aos clientes se mantenha alta.

Capítulo 2

Cloud Computing

A proposta desenvolvida é baseada em *cloud computing* e devido a isto é importante apresentar os componentes básicos que a compõe. Neste capítulo será introduzido os principais modelos de *cloud computing* que existem e seus componentes principais como gerência, tipos de armazenamentos e arquitetura de rede. Será apresentado também as diferentes formas de implementação dos conceitos abordados.

2.1 Visão Geral

Segundo o *National Institute od Standards and Technology* (NIST) [16] *cloud computing* é um ambiente, compartilhado ou não, com alta eficiência, automatizado e preferencialmente independente de *hardware* de infraestrutura de telecomunicações onde estes recursos podem ser provisionados sob demanda de qualquer lugar da rede podendo ser monitorado. O NIST [16] diz ainda que um modelo de *cloud computing* deve apresentar algumas características essencias descritas a seguir:

- **Auto Atendimento sob demanda:** funcionalidades computacionais são providas automaticamente sem a interação humana com o provedor de serviço.
- **Amplo acesso a serviços de rede:** recursos computacionais estão disponíveis através da Internet e são acessados via mecanismos padronizados, para que possam ser utilizados por dispositivos móveis e portáteis, computadores, etc.
- ***Pool* de recursosL:** recursos computacionais (físicos ou virtuais) do provedor são utilizados para servir a múltiplos usuários, sendo alocados e realocados dinamicamente conforme a demanda.

- **Elasticidade rápida:** as funcionalidades computacionais devem ser rápidas e elasticamente providas, assim como rapidamente liberadas. O usuário dos recursos deve ter a impressão de que ele possui recursos ilimitados, que podem ser adquiridos (comprados) em qualquer quantidade e a qualquer momento. Elasticidade tem três principais componentes: escalabilidade linear, utilização on-demand e pagamento por unidades consumidas de recursos.

- **Serviços mensuráveis:** Os sistemas de gerenciamento utilizados pela *cloud computing* controlam e monitoram 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 proveor de serviços, assim como para o consumidor do serviço utilizado.

A figura 2.1 ilustra a definição dada pelo NIST [16] para *cloud computing*.

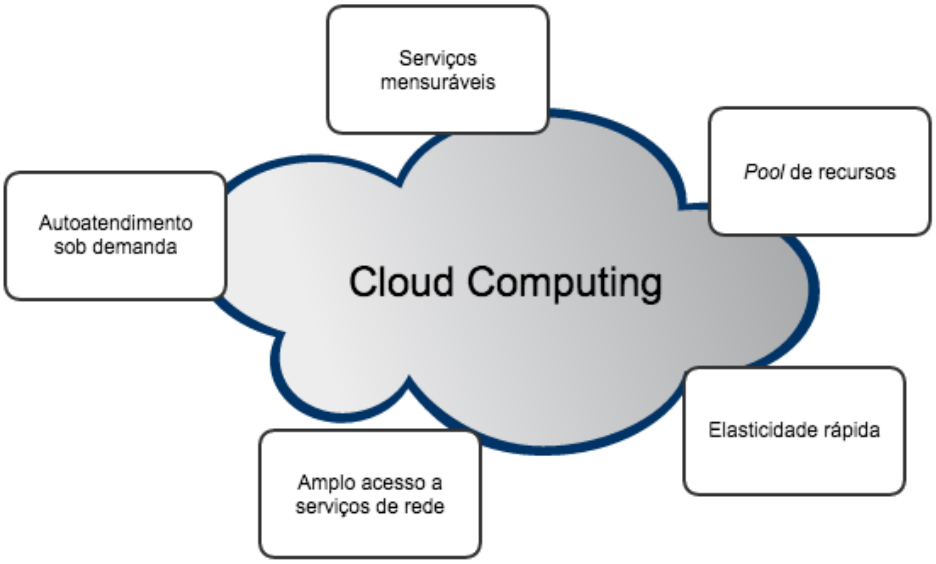


Figura 2.1: Características de *cloud computing* segundo NIST.

2.1.1 Modelos de Serviço

Segundo Manoel Veras [20] existem três principais modelos de serviços para *cloud computing*, São eles:

- **Infraestrutura como um serviço (*Infrastructure as a Service - IaaS*):** capacidade que o provedor tem de oferecer uma infraestrutura de processamento e

armazenamento de forma transparente. Neste cenário o usuário não tem o controle da infraestrutura física, mas , através de mecanismos de virtualização, possui controle sobre as máquinas virtuais, armazenamento, aplicativos instalados e possivelmente um controle limitado dos recursos de rede. Um exemplo de IaaS é a opção Amazon EC2.

- **Plataforma como um serviço (*Platform as a Service - PaaS*):** capacidade oferecida pelo provedor para o desenvolvimento de aplicativos que serão executados e disponibilizados na *cloud*. Este tipo de plataforma oferece um modelo de computação, armazenamento e comunicação para os aplicativos. Exemplos de PaaS são a AppEngine do Google e o Windows Azure da Microsoft.

- **Software como um serviço (*Software as a Service - SaaS*):** aplicativos de interesse para a grande quantidade de clientes passam a ser hospedados na *cloud* como uma alternativa ao processamento local. Os aplicativos são oferecidos como serviços por provedores e acessados pelos clientes por aplicações como o *browser*. Todo o controle e gerenciamento da rede, sistemas operacionais, servidores e armazenamento é feito pelo provedor de serviço. O Google Apps e o SalesForce.com são exemplos de SaaS.

A figura 2.2 ilustra a estrutura de modelos de serviços em *cloud computing*.

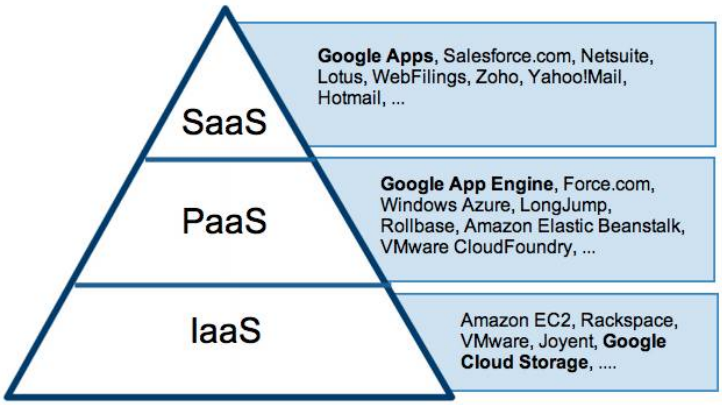


Figura 2.2: Estrutura de modelos de serviços em *cloud computing*.
Referência: [8]

2.2 Modelos de Implantação

Uma *cloud* pode ser implantada de várias maneiras. As quatro principais estão descritas a seguir:

2.2.1 *Cloud* Privada

Este tipo de implantação é destinado apenas para uma única empresa. Pode ser gerenciada internamente pela própria organização, ou terceiros. Neste modelo os serviços não estão disponíveis publicamente e o seu acesso se dá por tecnologias de transmissão com alta segurança onde os dados são criptografados na fonte e descriptografados somente em seu destino.

A *cloud* pública requer um nível significativo de envolvimento da gestão pois todo o ambiente de negócio deverá ser virtualizado tornando um desafio a integração com recursos já locais já existentes. A *cloud* privada proporciona um ambiente seguro, com elevada disponibilidade e tolerante a falhas, pontos que não são possíveis, como será visto a seguir, em uma *cloud* pública. Em contra-partida esta robustez pode resultar, a curto prazo, em um investimento elevado quando comparado a soluções de *cloud* públicas visto que alguns recursos como processamento, memória e armazenamento para alocação de dados são exclusivamente alocados e portanto se tornam mais caros.

Este modelo é idel para o uso em projetos de missão critica e que demandam confidencialidade dos dados armazenados e alocação exclusiva dos recursos de rede. Uma pesquisa da Forbes Insights em parceria com a Cisco [5] apontou que 52% das empresas privadas estão avaliando a adoção de nuvens privadas.

2.2.2 *Cloud* Pública

Em um modelo de *cloud* pública os serviços, as aplicações e o armazenamento são disponibilizados publicamente aos usuários geralmente através dos modelos PaaS e SaaS de serviço. Serviços popularmente conhecidos como os já citados Google App Engine e Windows Azure utilizam este modelo de implantação.

O modelo de negócio mais comumente adotado em uma *cloud* pública é o de pagamento conforme utilização ou *pas-as-you-go* em inglês onde o usário efetua o pagamento somente pelo recurso e tempo que utilizou. Esta mesma pesquisa da Forbes Insights [5] revelou que 38% das empresas estão avaliando a adoção de nuvens públicas e este interesse ocorre pois permite reduzir a complexidade da infraestrutura, reduzir os prazos de execução pois os recursos são escaláveis e também por oferecer uma economia de escala com o modelo de *pas-as-you-go*. No entanto este modelo proporciona menor flexibilidade para configurações particulares de segurança e desempenho uma vez que os recursos são compartilhados entre os usuários.

A *cloud* pública é ideal para usuários que precisam disponibilizar rapidamente sua aplicação no mercado ou que pretendam terceirizar sua infraestrutura de rede sem se preocupar com as questões regulamentares.

A Figura 2.3 ilustra a diferença entre *cloud* pública e privada do ponto de vista da forma de acesso.

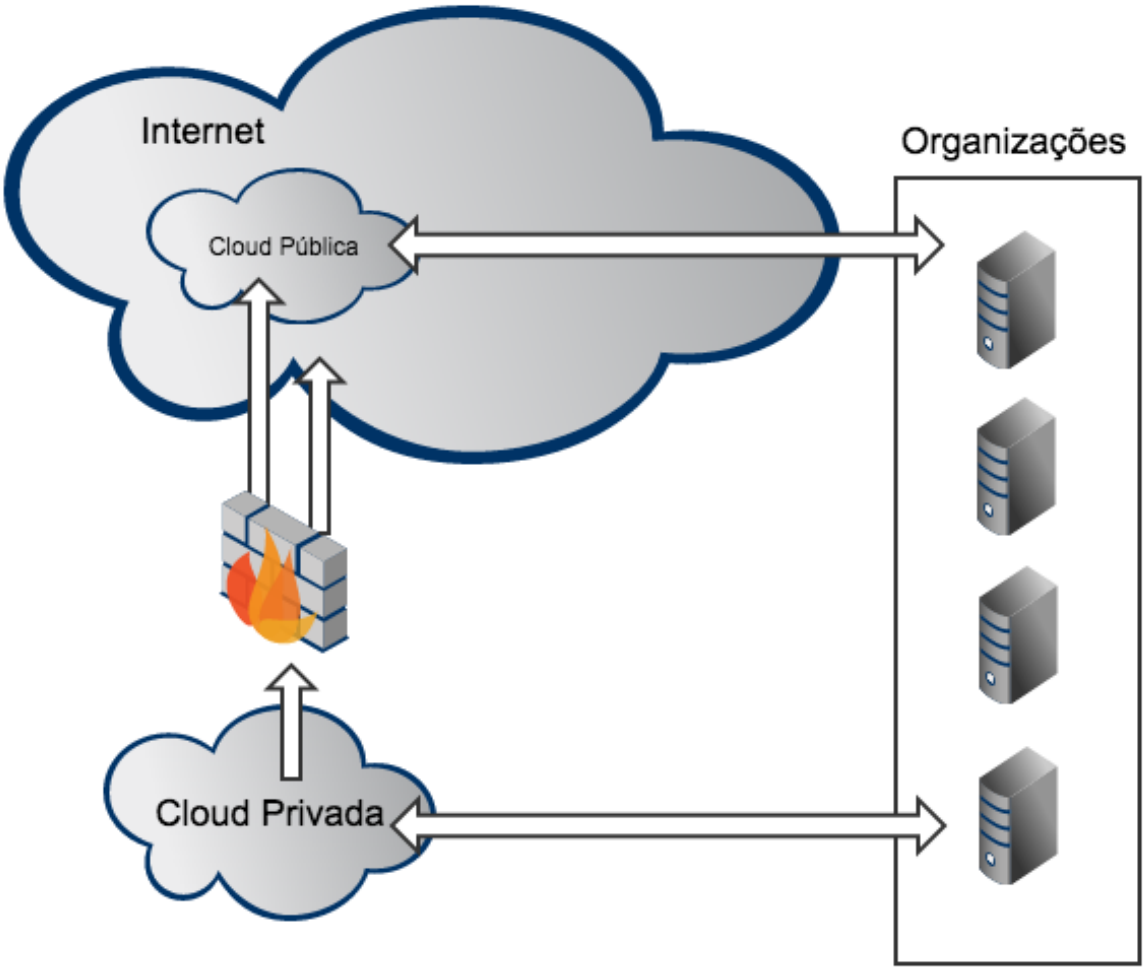


Figura 2.3: Visão esquemática de uma *cloud* Privada e Pública
[]

2.2.3 *Cloud* Híbrida

Como o próprio nome sugere uma *cloud* híbrida é constituída por serviços em *cloud* privada e também por serviços em *cloud* pública. Normalmente as empresas executam uma aplicação em sua *cloud* privada, mas utilizam a pública nos momentos de maior demanda de recursos. Para o melhor aproveitamento deste modelo é necessário que haja políticas de uso bem definidas além de regras de segurança para o acesso aos dados.

Um bom exemplo de uso de nuvens híbridas são os grandes comércios eletrônicos que precisam responder às flutuações de tráfego diário ou sazonal. Este tipo de mercado pode se beneficiar das características de elasticidade dos recursos da *cloud* pública. Contudo a regulamentação é rigorosa quando se trata de informações pessoais e de pagamento cabendo alocar este tipo de dado na *cloud* privada.

MELHORAR... INSERIR GRAFICOS DE ALOCACAO DE RECURSOS X DEMANDA.

2.3 *Cloud Computing* ou *Data Center* próprio

Um levantamento feito pela Global Technology Adoption Index (GTAI) [7] entrevistou 2 mil decisores de empresas de 11 países , entre eles cerca de 200 brasileiros, sobre seus hábitos e projetos envolvendo cloud. Este levantamento concluiu que 90% das médias empresas brasileiras consultadas afirmaram ter algum tipo de aplicação em cloud. Este percentual coloca o Brasil ao lado do México como mercado com maior penetração do conceito DE *cloud*. A média global de adesão é de 79% e menos de 1% dos entrevistados não consideraram projetos nesta área.

Outro estudo, este realizado pela Information Technology Industry Association (CompTIA) [6] entre os meses de junho e julho de 2014 revelou que mais de 90% das empresas de telecomunicação dos EUA estão usando alguma forma de *cloud computing*. Isto indica que a *cloud* se tornou de fato parte fundamental na nova infraestrutura de redes de comunicação nos dias de hoje.

A principal vantagem de se hospedar serviços e dados em um ambiente externo é o custo de um data center próprio. Instalar um data center TIER 4 , ou seja, com alta tolerância a falhas, e que siga a norma de infraestrutura American National Standards Institute / Telecommunications Industry Association (ANSI/TIA 942) [**?**] no Brasil custa em torno de US$60 milhões segundo pesquisa da FrostSullivan encomendada pela Brasscom [**?**]. Este valor pode chegar a US$43 milhões nos EUA.

Além do custo para instalação há os gastos mensais que , segundo a própria pesquisa, podem chegar a US$314 mil no Brasil e US$110 mil nos EUA somente com energia elétrica. Quando se considera todos os outros gastos como segurança, monitoramento , links de internet, gestão de rede, gestão de hardware, gestão de software, climatização, grupo motor gerador, backup de dados, manutenção civil, manutenção elétrica e sistema de combate a incêndio esses gastos podem chegar a US$950 mil mensais no Brasil e US$510

mil mensais nos EUA.

Portanto é fato que somente uma empresa com muitos recursos financeiros conseguirá arcar com todos estes custos sendo uma boa solução terceirizar infraestrutura contratando serviços baseados em *cloud computing* e mantendo assim o foco na qualidade de seus serviços.

Capítulo 3

Desafios relacionados a Alta Disponibilidade

3.1 Visão Geral

Alta Disponibilidade , segundo a definição do *Disaster Recovery Institute* () [13], pressupõe um conjunto de soluções ou aplicações que requerem um elevado nível de confiabilidade e disponibilidade, que operam 24 horas por dia, 7 dias por semana e usualmente requerem redundância intríseca para minimizar o risco de quedas devido a quaisquer problemas em hardware ou telecomunicações.

Segundo Fábio Lúcio Soares Gomes e Télio Luiz Pacheco [13] os requisitos para construção de um ambiente de alta disponibilidade, anteriormente definidos pela tecnologia da informação (TI), hoje são um dos produtos gerados pelo *Business Impact Analysis* (BIA).

Um ambiente de Alta Disponibilidade, TIER 4, é projetado para funcionar 24 horas por dia, 7 dias por semana, com total controle e integridade da infra-estrutura de Tecnologia da Informação e de Telecomunicações nele abrigadas, independente das variáveis externas. Alcança uma disponibilidade prevista de 99,995%, o que nos leva a um tempo máximo de indisponibilidade de 24 minutos anuais.

Desta forma qualquer solução que esteja utilizando este ambiente dependerá apenas da sua infraestrutura de hardware e software para implementar a disponibilidade necessária ao suporte do negócio.

A Figura 3.1 classifica a disponibilidade das soluções, implementadas em um Datacenter TIER 4:

- Aplicações Stand-Alone – possuem apenas um servidor e caso o mesmo esteja fora

de serviço, estima-se um tempo médio de recuperação de 87h36m;

- Aplicações com Duplicação de Servidores – possuem replicação de dados em outro servidor no próprio Datacenter, estima-se um tempo médio de recuperação de 8h45m;
- Aplicações com Diversidade Geográfica – possuem replicação em outro ambiente geograficamente distinto, sem possuir entretanto automatização para recuperação de desastres. O tempo estimado médio de recuperação de 0h52m;
- Ambiente de Alta Disponibilidade – mesma condição das soluções implementadas com Diversidade Geográfica com o diferencial de já terem automatizado todas as etapas. O tempo médio estimado para recuperação é da ordem de 0h5m.

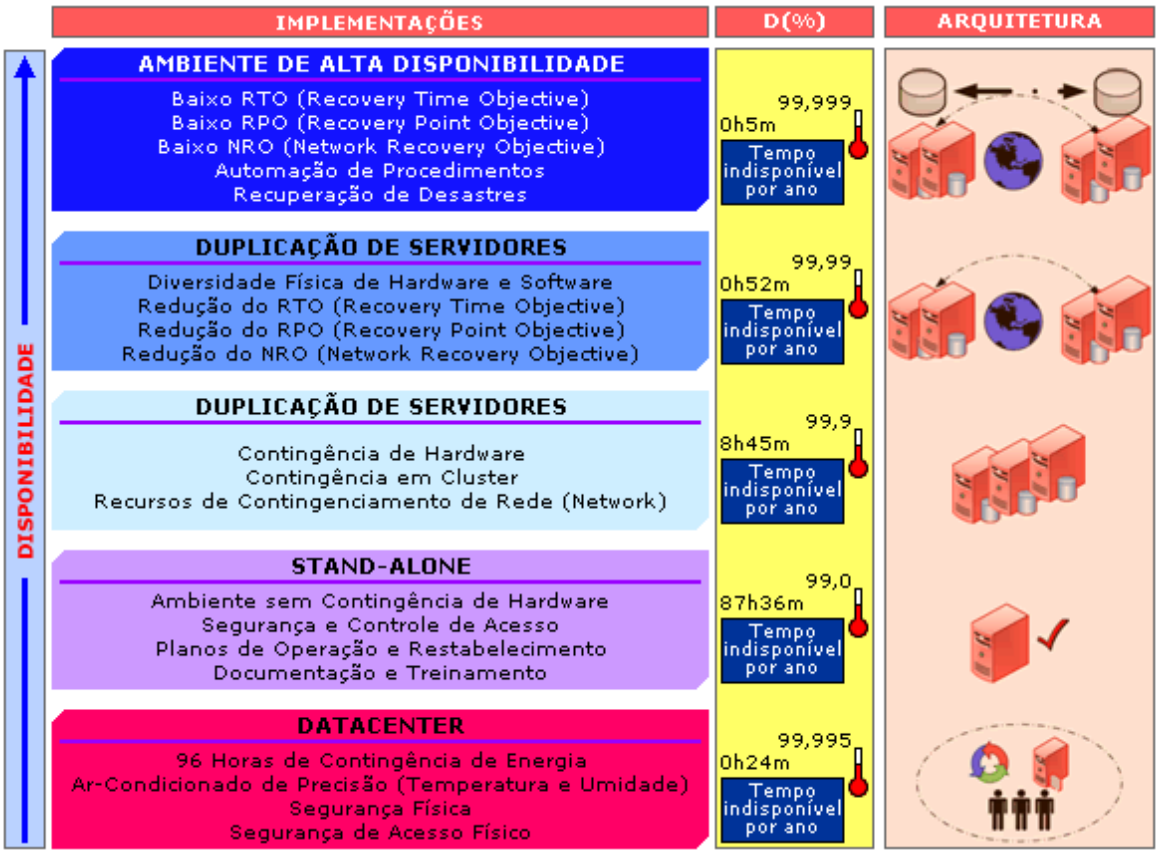


Figura 3.1: Topologia da arquitetura de backup. Referência: [13]

Figura 1: Níveis de Disponibilidade.

Os três parâmetros fundamentais para definição dos requisitos dos sistemas de missão crítica, gerados pelo BIA, são:

- RTO (Recovery Time Objective): Compreende o tempo máximo que o negócio pode suportar sem a solução tecnológica;
- RPO (Recovery Point Objective): Compreende o ponto de recuperação dos dados, ou seja, uma vez recuperada a solução qual a quantidade de dados máxima que poderá ser perdida sem que o negócio seja afetado;
- TRPO (Technical Recovery Point Objective): Compreende o tempo máximo para que todos os problemas técnicos sejam solucionados;
- NRO (Network Recovery Time): Compreende o tempo máximo de recuperação da rede.
- SLA (Service Level Agreement): Compreende em um acordo de nível de serviço entre cliente e fornecedor;

Os parâmetros RTO e RPO estão diretamente relacionados com as técnicas de replicação e backup de base de dados e com o valor que o dado representa para o negócio. Estes conceitos serão visto com mais detalhes no próximo capítulo.

A Figura 3.2 ilustra as diversas técnicas de backup e replicação necessárias, em função dos tempos de recuperação em caso de desastres. Observa-se pelo gráfico que técnicas que permitem uma recuperação quase imediata do dado (replicação assíncrona e síncrona) possuem um elevado custo para implementação e operação e se justificam apenas pelo valor que a informação representa para a empresa.

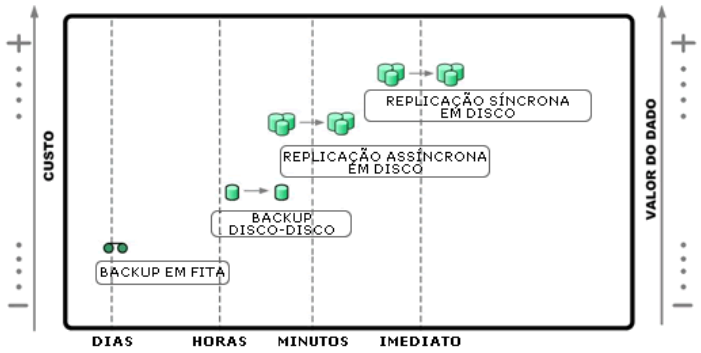


Figura 3.2: Tempo de recuperação de dados. Referência: [13]

A implementação de um ambiente de Alta Disponibilidade requer um complexo compromisso entre as necessidades do negócio, infra-estrutura do ambiente, arquitetura das soluções (hardware e software) para a garantia dos objetivos do BIA (RTO, RPO e NTO).

3.2 *Downtime*

Quando determinado serviço se torna indisponível por determinado tempo, independente do motivo, é usualmente dito que tal serviço enfrentou um *downtime* durante este período. Ou seja, as funções para o qual este serviço foi desenvolvido não estavam disponíveis. O desafio para toda empresa é reduzir o *downtime* de seus serviços a zero. Porém é sabido que tal métrica é muito difícil de ser alcançada.

3.2.1 Custos de *Downtime*

Existem duas categorias de custo de downtime, direto e indireto, que serão apresentados com mais detalhes nas seções abaixo.

3.2.1.1 Custos Diretos

Custos diretos são aqueles que podem ser medidos pelo departamento financeiro da empresa que usa o serviço que sofreu um desastre. São eles:

- Perda de produtividade do colaborador: Inclui o tempo perdido (e pago) ao colaborador da empresa que usa o sistema inoperante;
- Perda de oportunidades: A perda de oportunidade é a perda da chance realizar negócios tais como o provisionamento de um novo cliente na rede.
- Custos regulatórios e indenizações: Estes custos incluem taxas e multas devido a violaçõe de regulações de leis governamentais tais como o não envio do protocolo de utilização de determinado serviço via SMS ou retorno automático da chamada pelo atendente de call center após a chamada original ser interrompida. Estas duas medidas, por exemplo, são imposições da ANATEL (Agência Nacional de Telecomunicações) as operadoras de telefonia móvel no Brasil.

Uma pesquisa feita pelo instituto Gartner analisou o custo por hora médio para alguns setores devido a tempo de inatividade de seus serviços como pode ser visto na tabela abaixo:

Custo médio de downtime por hora (US$)	
Atividade/Setor	US$/hora
Bolsa de valores	6.5 milhões
Setor energético	2.8 milhões
Cartão de crédito	2.6 milhões
Telecomunicações	2.0 milhões
Bancos	1.5 milhão
Indústria	1.6 milhão
Varejo	1.1 milhão
Setor farmacêutico	1.0 milhão
Indústria Química	704 mil
Saúde	636 mil
Setor de transporte aéreo	340 mil

Tabela 3.1: Custo médio por hora de inatividade - Adaptado de [9]

3.2.1.2 Custos Indiretos

Custos indiretos são difíceis de mensurar porém podem ter grande impacto e pode se estender por um longo período de tempo. Alguns exemplos são:

- Satisfação do cliente: Se um cliente não conseguir realizar uma ligação ou enviar uma mensagem imediatamente ele poderá ser incentivado a procurar os serviços de um concorrente.
- Má publicidade: O serviço inativo pode ter repercussão negativa na mídia o que degrini a imagem da empresa.
- Preço das ações: Uma consequência comum da má reputação devido a um serviço inativo é a queda do preço das ações da empresa. Por exemplo uma região inteira sem internet devido a queda de um sistema de gerenciamento de rede.
- Responsabilidade Legal: Um serviço fora do ar pode gerar multas. Em casos extremos executivos podem ser responsabilizados por negligência.
- Moral dos funcionários: Uma má reputação da empresa pode causar um mal estar dos próprios funcionários-chave culminando com a sua saída de empresa.

Capítulo 4

Disaster Recovery

Disaster Recovery (DR) trata-se das ações a serem tomadas para reestabelecer determinado sistema ou serviço aos níveis aceitáveis de operação garantindo assim a sua qualidade. Como dito anteriormente estamos cada vez mais dependentes dos sistemas de telecomunicações e a inatividade destes sistemas podem gerar problemas. Muitas empresas e até mesmo governos utilizam sistemas de recuperação de desastres (DR) para minimizar o seu tempo de inatividade. Há diversos mecanismos de recuperação de desastres atualmente como backups periódicos em fita ou replicação síncrona do banco de dados entre sistemas geograficamente separados. Um dos principais desafios na prestação de serviços de DR é o chamado *Business Continuity* (BC) permitindo que as aplicações possam rapidamente ficar acessíveis após um período de falha o que não é muito simples de se conseguir quando se usa os mecaninsmos citados anteriormente. Minimizando o tempo de recuperação e os dados perdidos devido a um desatre, um serviço de DR pode até prover um BC porém a um alto custo. A idéia do projeto é oferecer um sistema capaz de monitorar e viabilizar a rápida recuperação a desastres.

4.1 Visão Geral

A computação na nuvem baseada em virtualização gerou uma abordagem muito diferente para recupeção de desastres. Com a virtualização todo o servidor, incluindo o sistema operacional, aplicações e dados são encapsulados em um único pacote. Esse pacote pode ser copiado para um qualquer destino, incluindo um data center *offsite* em questão de minutos. Um vez que o servidor virtual é independente do hardware todo o pacote (sistema operacional, aplicações e dados) podem ser transferidos de um lugar para outro diminuindo drasticamente o tempo de recuperação quando comparado com

o modo tradicional (não virtualizado) onde os servidores precisam ser iniciados com a última configuração usada antes que os dados possam ser restaurados.

Quando consideramos a restauração online usando a computação na nuvem os métodos tradicionais de backup não fazem mais sentido e torna-se difíceis de serem justificados quando considerados a relação custo-benefício e o tempo de restauração. A computação na nuvem torna a recuperação a desastres a frio ou cold site disaster recovery (CSDR) antiquada já que temos, por exemplo, a arquitetura warm site disaster recovery (que será discutida mais a frente) como uma alternativa mais econômica um vez que sistemas e serviços críticos podem ser migrados para uma nuvem privada ou pública em questão de segundos sem interrupção do serviço , o que chamamos de hot disaster recovery (HSDR).

Um dos grandes benefícios do uso da computação na nuvem para a recuperação de desastres é a capacidade de dimensionamento do uso dos recursos disponíveis e desempenho desejados. Aplicações e serviços que são considerados menos críticos em um eventual desastre podem ser despriorizados e receberem menos recursos como processador, memória ou carga garantindo que as aplicações e serviços mais críticos recebam os recursos necessários para o funcionamento adequado.

A migração das aplicações e serviços podem ser feitos também a nível de camada de rede. Há opções de migração também dos mapeamentos IPs, regras de firewall e configurações de VLAN para a nuvem desejada. Isto possibilita não somente a migração dos software e dados mas também toda a estrutura de rede acerca da aplicação ou negócio.

Neste contexto podemos definir desastre como qualquer impacto negativo sobre a continuidade operacional ou financeira do negócio. A causa pode estar entre as mais variadas:

- falha de hardware
- falha de software
- falha de rede
- falha de energia
- dados físicos ao site como:
  - incêndio
  - inundação

- erro humano
- sabotagem

E as consequências também podem ser diversas, como:

- Perda de dados importantes para o negócio, por exemplo:
  - transações bancárias
  - operação de compra ou venda
  - interrupção de uma chamada telefônica
- Degradação da reputação da empresa
- Perdas financeiras

Para minimizar o impacto de um desastre no negócio, as empresas investem tempo e recursos para planejar, preparar e implementar processos de atualização para lidar com esses eventos. O valor do investimento para o planejamento de recuperação de desastres de um sistema espeçifico pode variar dependendo do custo de uma parada. Descreveremos algumas abordagens típicas que vão desde investimentos mínimos a disponibilidade em larga escala e tolerância a falhas.

4.2 Conceitos Básicos

Introduziremos com mais detalhes os três conceitos importantes quando se fala de recuperação a desastres. São eles:

- Recovery Time Objective
- Technical Recovery Point Objective
- Recovery Point Objective
- Network Recovery Objective
- Service Level Agreement

O Recovery Time Objective (RTO) é o tempo de duração e o nível de serviço em que uma aplicação precisa ser restaurada após um desastre (ou interrupção) para garantir parâmetros aceitáveis de continuidade do negócio. Por exemplo, se um desastre ocorrer às 11:00 AM e o RTO do sistema for de 9 horas, o processo de DR precisa garantir uma recuperação a níveis aceitáveis de serviço até às 20:00 PM. Do conceito de RTO deriva-se o parâmetro que chamamos de Technical Recovery Time Objetive (TRTO), medido também em horas , que consiste no tempo limite para que todos os problemas técnicos sejam solucionados. Portanto o TRTO deverá ser sempre menor que o RTO. Geralmente trabalha-se com a relação TRTO = RTO - 1 hora .

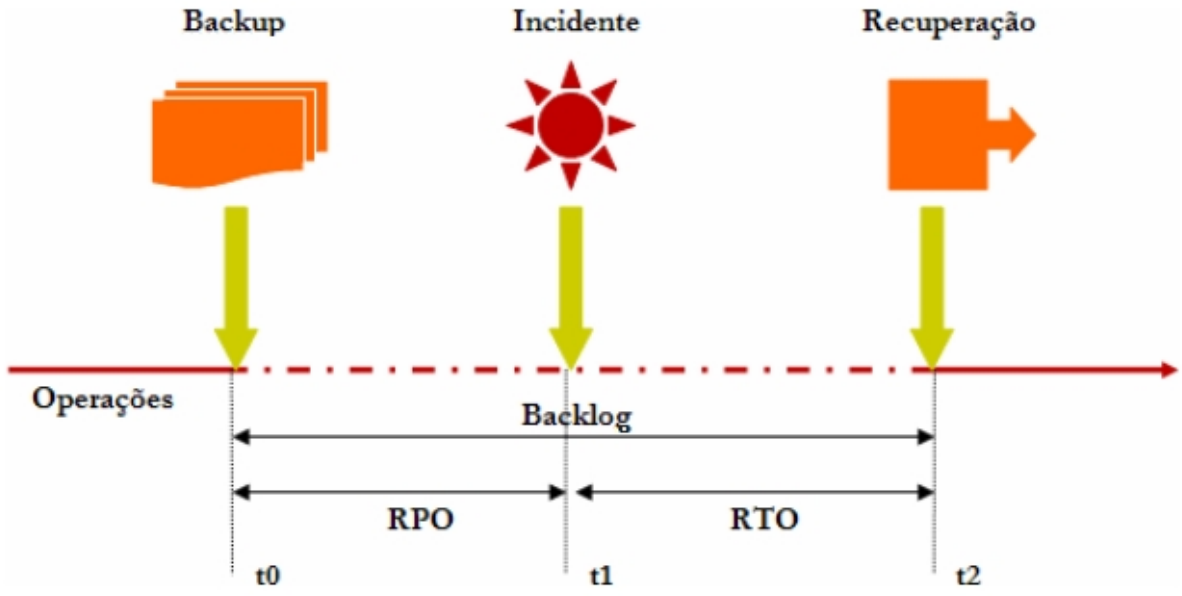


Figura 4.1: Ilustração dos conceitos de RPO e RTO.

O Recovery Point Objective (RPO). Este conceito descreve a quantidade aceitável de dados perdidos, medidos em tempo. Seguindo o mesmo exemplo anteriro, se o RPO do sistema for de 1 hora o DR terá que garantir, após o restabelecimento do sistema, a integridade dos dados até às 10:00 AM devido ao problema ocorrido às 11:00 AM.

O Network Recovery Objective é um conceito relacionado ao RTO no qual indica o tempo máximo que uma sistema pode ficar inacessível ou em um estado não funcional devido a uma falha em um elemento de rede. Este elemento pode ser, por exemplo, um roteador, switch, link ou mesmo um servidor DNS (Domain Name Server).

O Service Level Agreement trata-se de um acordo de nível de servíço criado entre o fornecedor e o cliente no qual é garantido um certo nível de serviço pré acordado entre as partes. Por exemplo, determinado enlace rádio tem um SLA 99,95% , ou seja, o fornecedor garanti que em 99,95% do tempo o enlace estará operacional.

Normalmente um empresa decide os parâmetros aceitáveis de RTO e RPO com base no impacto financeiro da não disponibilidade dos seus sistemas ou serviços. O impacto financeiro é avalidado considerando vários fatores entre eles perda de negócios e danos à reputação devido a indisponibilidade. Com base nisso as organizações planejam soluções para fornecer de forma econômica a recuperação do sistema baseado no RPO dentro do cronograma e um nível de serviço estabelecido pela RTO.

4.3 Práticas Tradicionais de Investimento em DR

Uma abordagem tradicional para DR envolve diferentes níveis de replicação de dados e de infra-estrutura. A redundância de sistemas críticos são mantidas em uma infra-estrutura replicada e testadas regularmentes. Essas redundâncias deve estar separadas suficientemente da infra-estrutura original para garantir isolamento total aos possíveis desastres que ocorram.

A infra-estrutura original e a redundante precisam oferecer serviços básicos para operação que seriam:

- Energia e refrigeração do ambiente.
- Proteção contra acesso físico indevido.
- Espaço para escalonamento.
- Suporte a reparos e trocas de equipamentos ou instalações.
- Contrato de serviço de Internet com banda necessária para os horários de pico.
- Elementos de rede como firewall, roteadores, switches e balanceadores de carga.
- Espaço suficiente para serviços de missão crítica como storage e aplicações de backend como autenticação de usuário, Domain Name Server (DNS), Dynamic Host Configuration Protocol (DHCP) , monitoramento e alertas.

Dependendo da criticidade dos serviços caso o ambiente seja tolerante a falhas todos os itens acima deverão ser duplicados.

4.4 Arquiteturas Possíveis

4.4.1 Introdução à Amazon AWS

A Amazon é uma empresa americana muito conhecida pela sua loja virtual, inicialmente com venda de livros, hoje com a venda de diversos produtos, e por seu ótimo relacionamento com o cliente.

Há alguns anos atrás a Amazon passou por uma re-estruturação de toda a sua infra-estrutura de servidores e redes devido ao rápido crescimento de sua loja virtual. No final de 2003, Chris Pinkham e Benjamin Black publicaram um artigo demonstrando a padronização da infra-estrutura da Amazon, mencionando a possibilidade de venda de servidores virtuais como um serviço.

Com o sucesso desse projeto e da publicação do artigo, abriu-se uma grande oportunidade de vender serviços de *Cloud Computing*, e então iniciou-se o projeto da Amazon AWS.

O serviço foi efetivamente lançado em 2006, ainda com uma quantidade limitada de serviços que podem ser acessados através de HTTP, arquitetura REST e SOAP. Todos os serviços são baseados em cobrança por utilização, modelo *pay-as-you-go*. Os valores de cobrança são diferenciados para cada tipo de serviço.

Hoje, uma grande quantidade de serviços está disponível para contratação e estes podem ser hospedados em diversas regiões diferentes pelo mundo. Um cenário perfeito quando pensamos em alta disponibilidade.

Relatórios publicados pela Pacific Crest Securities apontam a Amazon AWS como o serviço que cresceu mais rápido na história da indústria. Esses estudos apontam que a receita da AWS estará na faixa dos US$ 5 bilhões neste ano e US$6,7 bilhões em 2015.

A figura 4.2 mostra o crescimento da Amazon AWS comparado ao de outras grandes empresas de tecnolgia.

Os serviços oferecidos pela AWS podem ser controlados através de um painel administrativo na internet ou até mesmo através da API do serviço.

Ao efetuar o login na Amazon AWS, o painel exibido na figura 4.3 é apresentado, onde se tem listado diversos serviços disponíveis para uma determinada região.

As regiões disponíveis são:

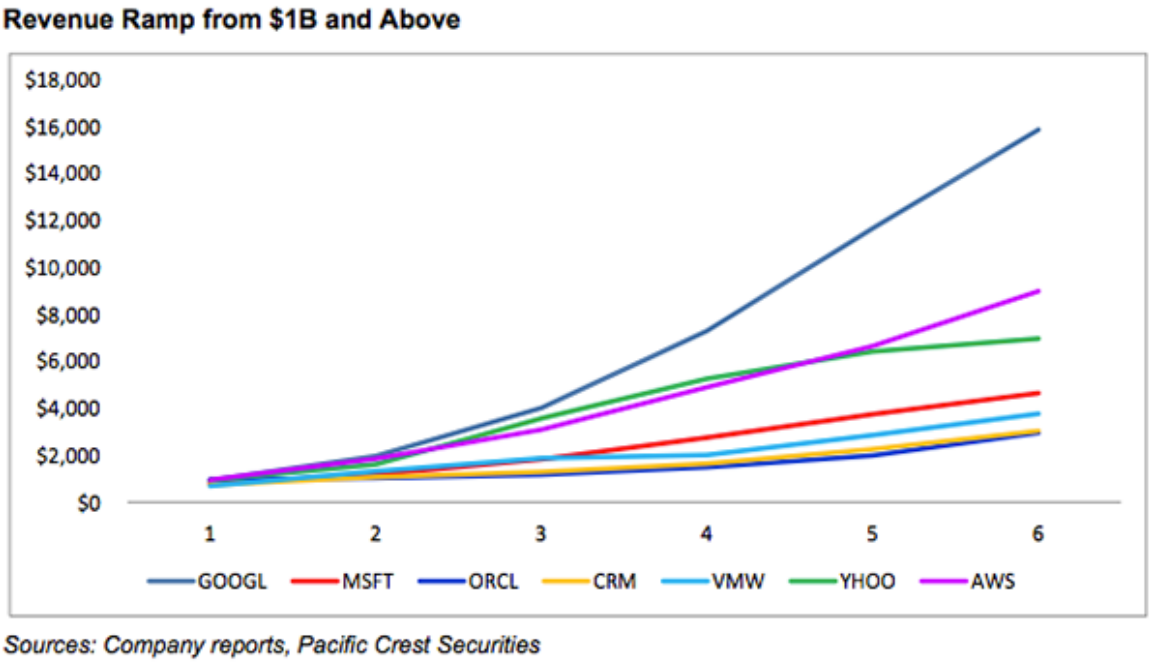


Figura 4.2: Gráfico ilustrando a receita ao longo do tempo para diversas empresas de tecnologia

1. North Virginia
2. Oregon
3. North California
4. Ireland
5. Frankfurt
6. Singapore
7. Tokyo
8. Sydney
9. São Paulo

No caso da figura 4.3, a região selecionada é a North Virginia. Os custos são variáveis para cada região, sendo os custos na North Virginia um dos mais baixos disponíveis e os custos em São Paulo os mais altos disponíveis.

As tabelas de custos 4.4 e 4.5 representando o serviço de máquinas virtuais (Amazon EC2) utlizando sistema operacional Linux para as regiões North Virginia e São Paulo são apresentadas abaixo:

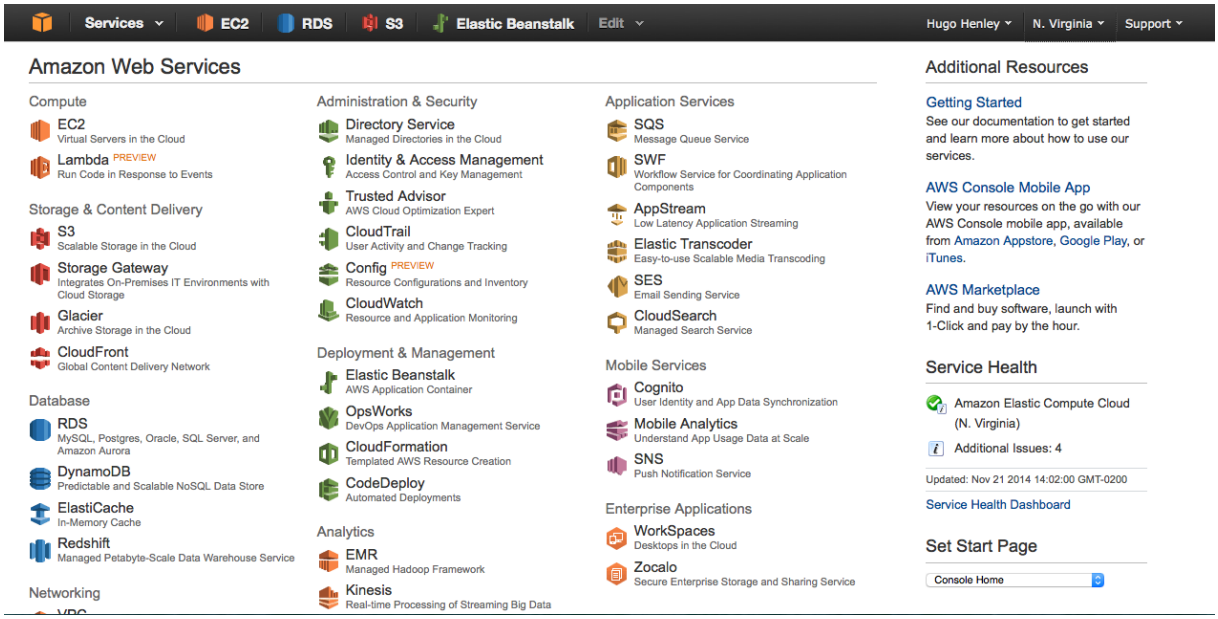


Figura 4.3: Painel de Administração da Amazon AWS

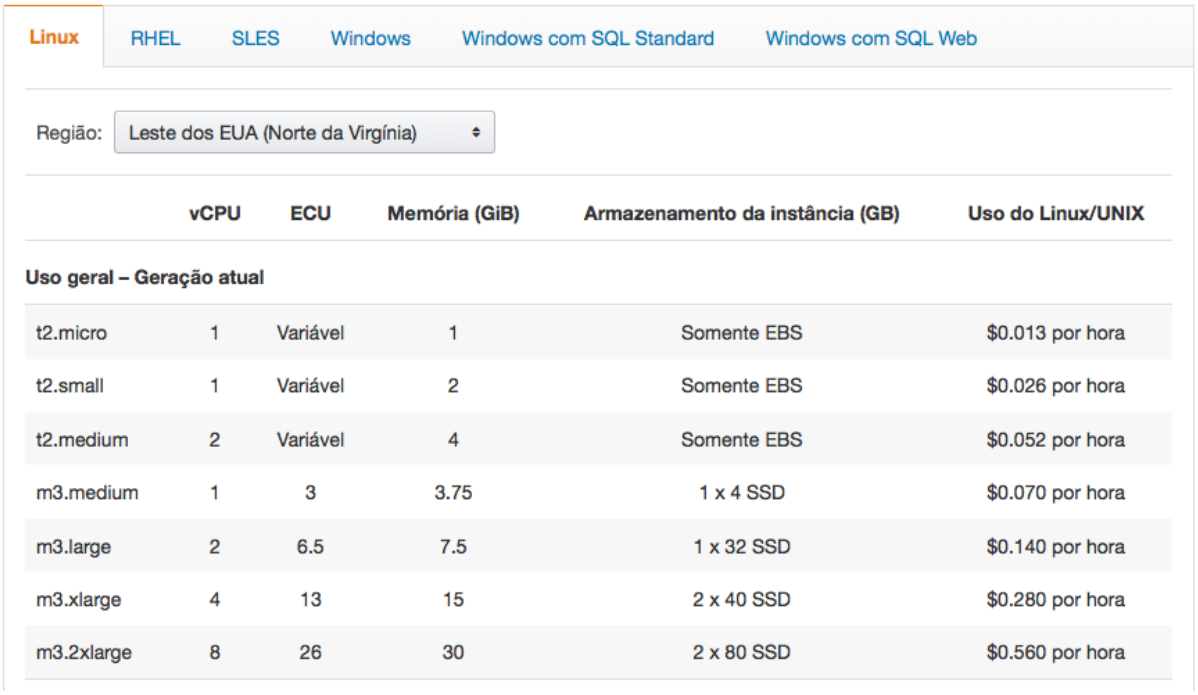
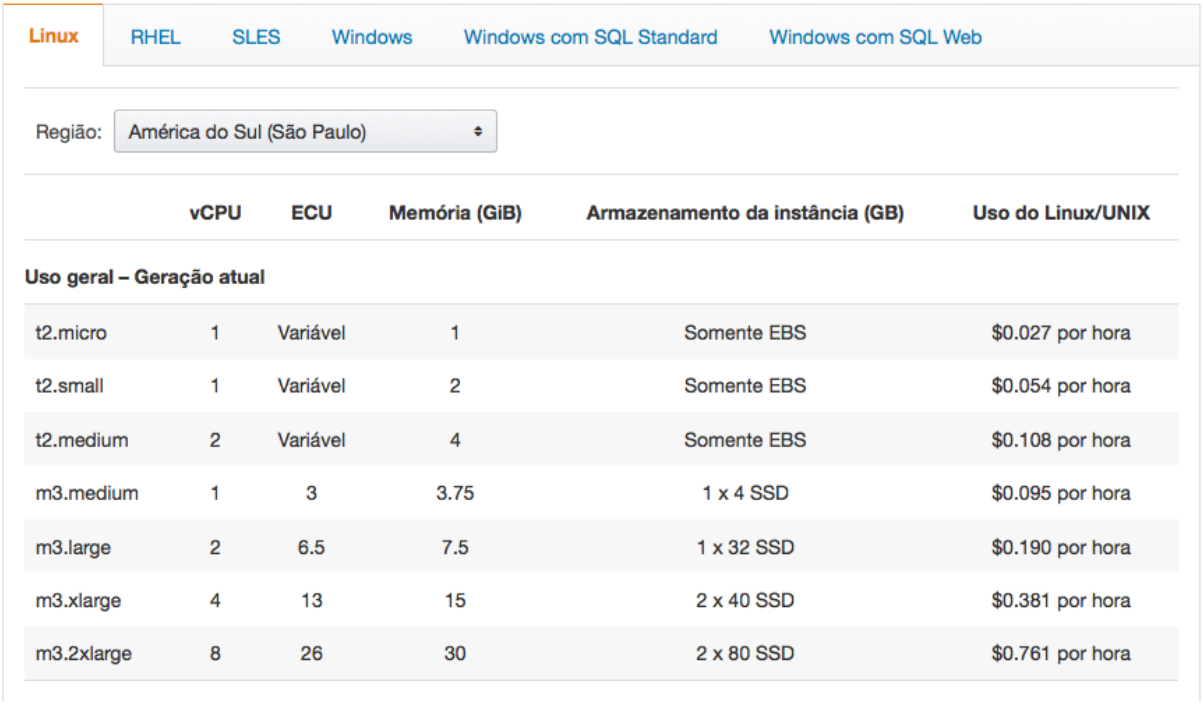


Figura 4.4: Tabela exibindo os custos do serviço EC2 no Norte da Virgínia http://aws.amazon.com/pt/ec2/pricing/

Muitos serviços fazem um paralelo com tecnologias existentes dentro de um Data Center. Os serviços S3 e Glacier funcionam como *Storage*, o EC2 como serviço de servidores virtuais, o RDS como serviço de gerenciamento de Banco de Dados Relacionais, o Elastic Beanstalk como serviço de gerenciamento e *deploy* de aplicações, o Route 53 para DNS e assim por diante. Os serviços contém uma boa documentação que pode ser lida



Linux | RHEL | SLES | Windows | Windows com SQL Standard | Windows com SQL Web

Região: América do Sul (São Paulo)

	vCPU	ECU	Memória (GiB)	Armazenamento da instância (GB)	Uso do Linux/UNIX
Uso geral – Geração atual					
t2.micro	1	Variável	1	Somente EBS	$0.027 por hora
t2.small	1	Variável	2	Somente EBS	$0.054 por hora
t2.medium	2	Variável	4	Somente EBS	$0.108 por hora
m3.medium	1	3	3.75	1 x 4 SSD	$0.095 por hora
m3.large	2	6.5	7.5	1 x 32 SSD	$0.190 por hora
m3.xlarge	4	13	15	2 x 40 SSD	$0.381 por hora
m3.2xlarge	8	26	30	2 x 80 SSD	$0.761 por hora

Figura 4.5: Tabela exibindo os custos do serviço EC2 em São Paulo
http://aws.amazon.com/pt/ec2/pricing/

gratuitamente no site da Amazon AWS. Recomenda-se a leitura para melhor uso destes pois uma decisão arquitetural errada para sua solução em *cloud* pode levar a um custo financeiro adicional desnecessário.

Depois desta breve apresentação sobre os serviços, é possível iniciar com maior detalhamento as arquiteturas de *Disaster Recovery* recomendadas pela Amazon AWS [11] .

4.4.2 Backup and Restore

A arquitetura de Backup e Restore é baseada em cópia dos principais arquivos e aplicações para uma fita ou disco rígido de alta capacidade (*storage*) periodicamente. Estes dados armazenados ficam disponíveis para restauração quando necessário. Há várias formas de implementar este conceito em uma empresa. Alguns optam por ter a gerência completa do backup e restauração. Para isso possuem uma estrutura física própria (servidores, storage ou robôs de fita) para viabilização do backup. Neste caso haverá a necessidade da criação e adoção de uma política de retenção e reuso da fita. Devemos lembrar que uma tem vida útil em torno de 20 anos, porém alguns tipos de dados devem ser mantidos por pelo menos 5 anos de acordo com a necessidade do negócio e cumpri-

mento de leis. Na figura 4.6 podemos ver esquematicamente o processo de backup em fita.

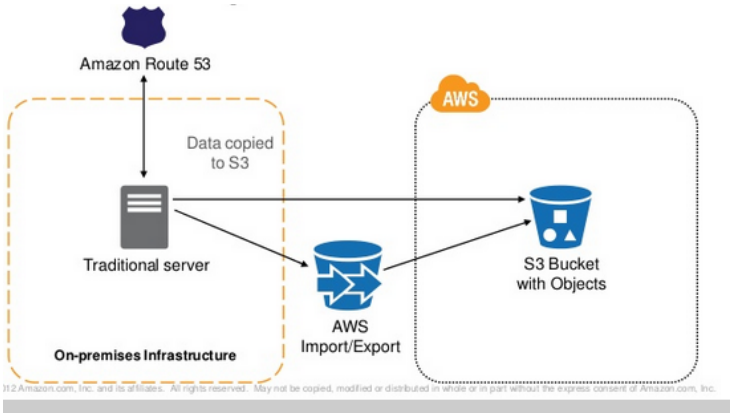


Figura 4.6: Topologia da arquitetura de backup
Referência: [11]

Outra forma de se implementar este conceito é terceirizando a mecânica de backup transferindo a responsabilidade de gerenciar, armazenar e disponibilizar os dados armazenados, para uma empresa parceira. A escolha de qual forma adotar depende da importância dos dados (sigilosos ou não) e a agilidade necessária no processo de restauração. A figura 4.7 ilustra este processo de restore.

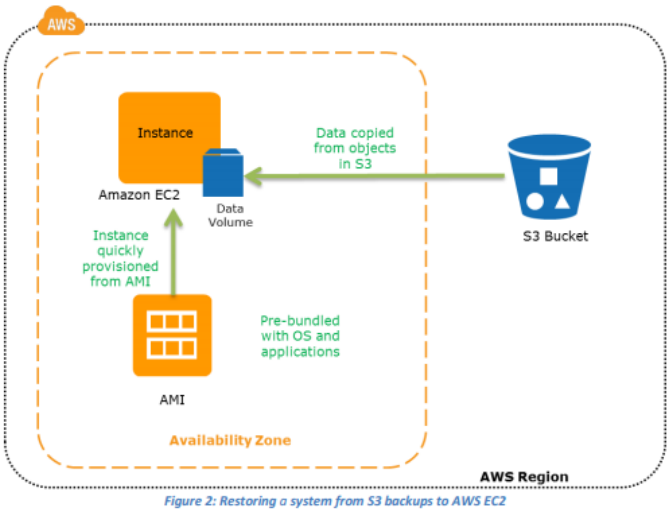


Figura 4.7: Topologia de restore usando backup
Referência: [11]

4.4.3 Pilot Light

Enquanto o conceito de Backup e Restore concentra-se principalmente nos dados a serem armazenados, o modelo Pilot-Light adiciona a preocupação com a camada de

aplicação. Este modelo provisiona um ambiente idêntico ao contido no Data Center e mantêm-se pronto para uso no momento de ocorrência de desastre. Diferente de uma solução que faz a replicação completa de todas as camadas do ambiente, o Pilot-Light replica apenas a camada de Banco de Dados, mantendo as outras camadas (*Application Server e WebServer*) sem a necessidade de sincronia.

As figuras 4.8 , 4.9 e 4.10 ilustram o modelo Pilot Light antes, durante e depois do desastre respectivamente.

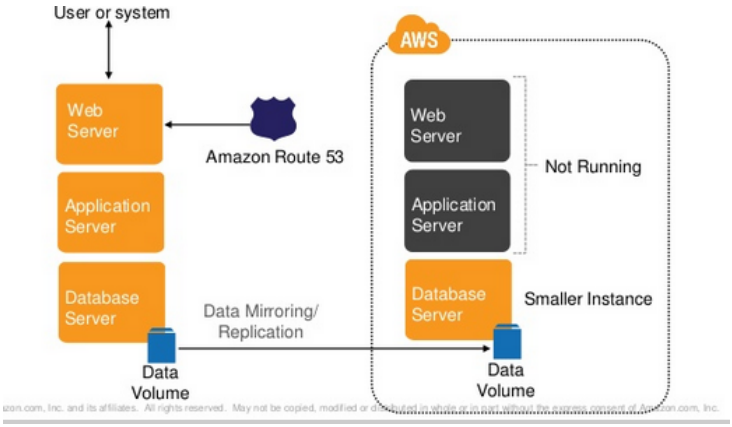


Figura 4.8: Arquitetura - Pilot Light antes do desastre
Referência: [11]

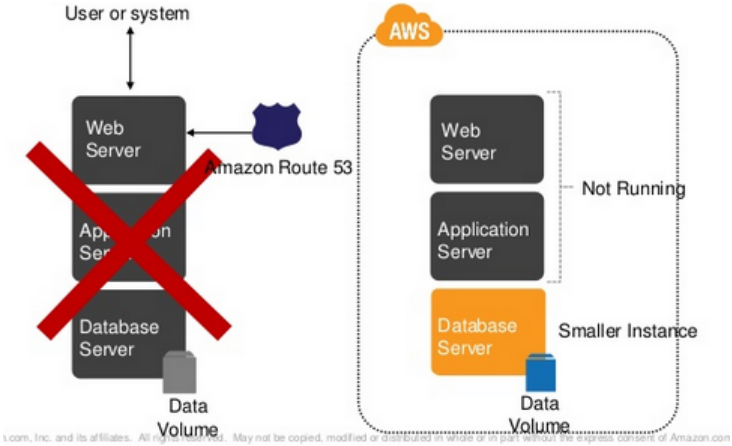


Figura 4.9: Arquitetura - Pilot Light durante o desastre
Referência: [11]

Esse é um modelo de baixo-custo, visto que no modelo *pay-as-you-go* o custo é variável de acordo com o que está sendo executado. Uma desvantagem que se pode notar é que, no momento em que ocorre o desastre, é preciso iniciar grande parte do ambiente, e essa operação pode se tornar demorada, não sendo adequada quando o DR deve ser executado em um tempo muito curto, na ordem dos milisegundos ou segundos.

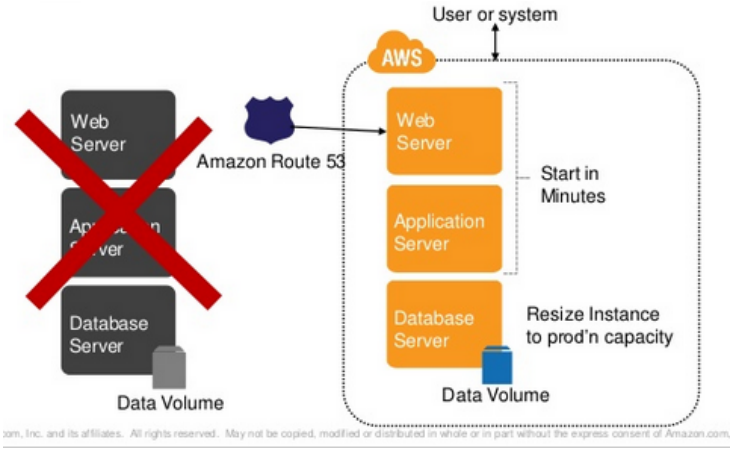


Figura 4.10: Arquitetura - Pilot Light após o desastre
Referência: [11]

Apesar dessa desvantagem, a solução desenvolvida em software para o problema relacionado ao DR apresentado neste trabalho é baseada neste modelo. Por ser um modelo de baixo-custo, tem-se a vantagem de conseguir maior *market share* para adoção. Além disso, muitas empresas ainda toleram, em momentos de falha, alguns minutos de *downtime*, principalmente quando há garantia de que uma vez recuperado, o ambiente tem grande probabilidade de não apresentar instabilidades (característica de escalabilidade possível com a utlização de *Cloud Computing*).

Estudos realizados pela empresa Accenture para produção do guia entitulado como "Disaster Recovery with Amazon WebServices: A Technical Guide"apontam esse modelo como indicado para grande parte das empresas que tem como requisito o backup contínuo, recuperação relativamente rápida e redundante. Essa solução é compatível com requisitos de baixo RPO e RTO moderado. De acordo com o guia, um RTO moderado compreende valores menores do que 4 horas para recuperação.

4.4.4 Warm Standby

Quando o DR é configurado utilizando a solução chamada *Warm Standby*, é possível de se ter uma solução com *downtime* próximo de zero, o que estaria de acordo com um *uptime* próximo de 100% exigido em alguns acordos de SLA.

Essa diferença é caracterizada pela arquitetura proposta nesse modelo. Enquanto no *Pilot-light* apenas algumas camadas, como a de Banco de Dados, eram replicadas, no *Warm Standby* há replicação também das outras camadas, como por exemplo a de aplicação. Sendo assim, o tempo gasto na solução anterior para que o ambiente esteja

pronto para produção não é mais necessário. Se a aplicação principal é interrompida em um evento de desastre, basta que o DNS seja redirecionado para o ambiente replicado e, em questão de segundos o ambiente responde sem que o usuário note diferença.

Não é difícil notar que apesar do menor tempo de *dowtime* alcançado, o custo dessa solução é mais elevado. Para diminuir um pouco esse custo, o ambiente replicado pode, quando em standby, não funcionar com a mesma especificação de máquina virtual em termos de recursos (número de núcleos de CPU, quantidade de memória RAM e HD/SSD). Dessa forma, no momento em que fosse necessário realizar a recuperação, esses recursos poderiam sofrer aumento automático para que estejam no padrão exigido para ambiente de produção.

As figuras abaixo exibem o modelo proposto pela Amazon para DR utilizando *Warm Standby* nas fases de preparação (figura 4.11) e recuperação (figura 4.12).

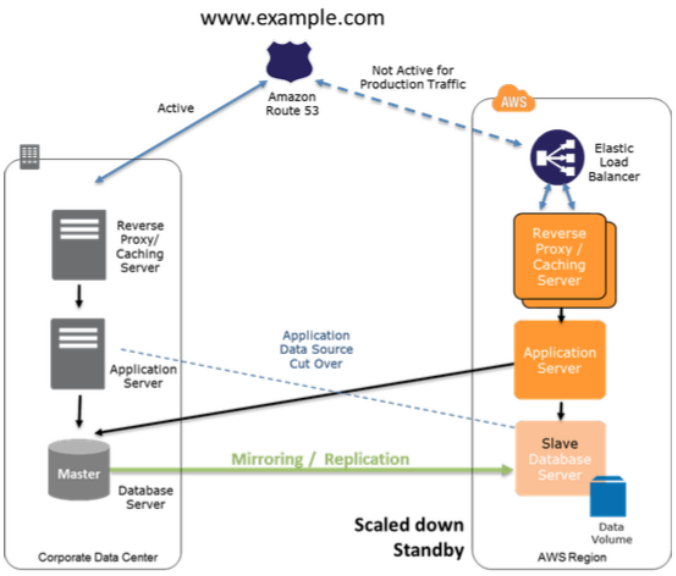


Figura 4.11: Fase de Preparação do *Warm Standby*
Referência: [11]

4.4.5 Multi-site Solution

Uma última opção recomendada pela Amazon é que se utilize Multi-site Solution. Nessa solução, o ambiente replicado em nuvem tem capacidade igual a do ambiente de produção no Data Center. Essa configuração é chamada de ativa-ativa. O tipo de replicação de dados escolhido é que vai determinar o RPO da solução.

Todo o tráfego recebido é balanceado entre os dois ambientes (físico e *cloud*). Sendo

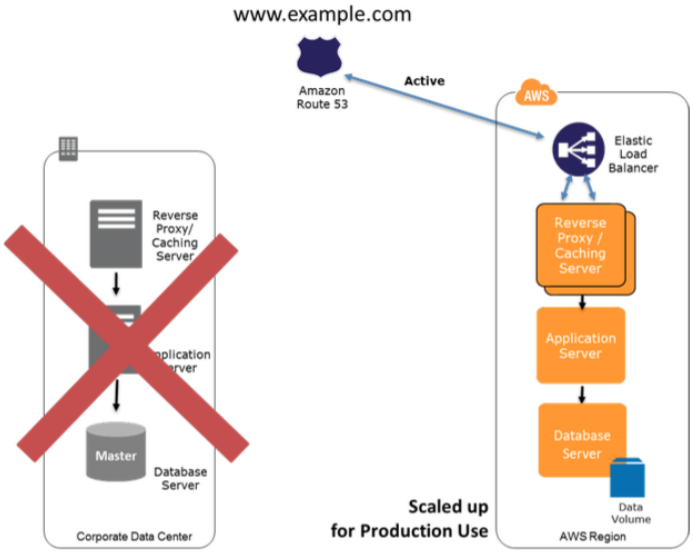


Figura 4.12: Fase de Recuperação do *Warm Standby*
Referência: [11]

assim, o ambiente em *cloud* sempre recebe requisições. No momento de desastre, a única alteração necessária ocorre no balanceador de carga. O serviço em Amazon responsável por DNS com capacidade de balanceamento é o *Route 53*.

Essa é a solução de DR mais cara que se pode obter, porém a com menor RTO e RPO, sendo então a mais eficiente. Uma análise financeira deve ser efetuada antes de se optar por esta solução, que pode apresentar alto custo caso o número de aplicação que terão um plano de DR seja alto e/ou seja exigido grande poder computacional.

As fases de preparação e recuperação estão ilustradas nas figuras 4.13 e 4.14 abaixo:

4.5 Impacto Financeiro

Para que se entenda os benefícios de utilizar uma solução de DR em *Cloud Computing*, é necessário fazer uma comparação de custos entre essa solução e outras disponíveis. Opções comuns para DR são:

1. Construir outro Data Center localizado a centenas de quilômetros do Data Center principal e com recurso computacional compatível
2. Alugar espaço em outro Data Center, prática conhecida como *Colocation*
3. Utilizar *Cloud Computing*

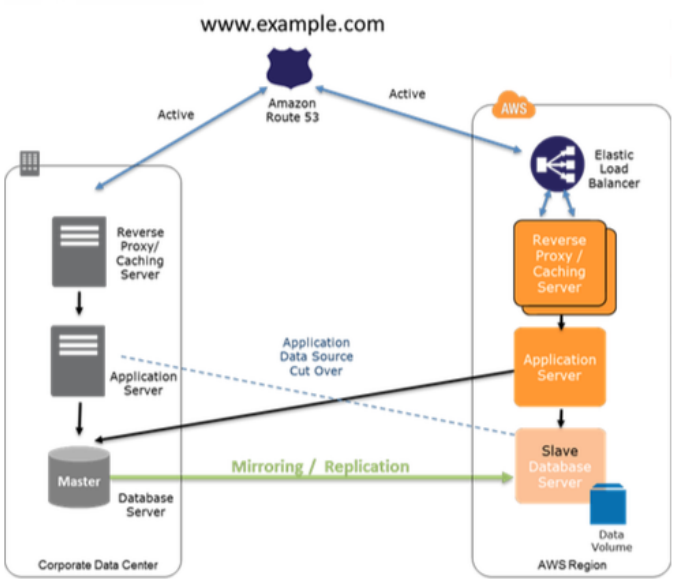


Figura 4.13: Fase de Preparação do *Multi-site Solution*
Referência: [11]

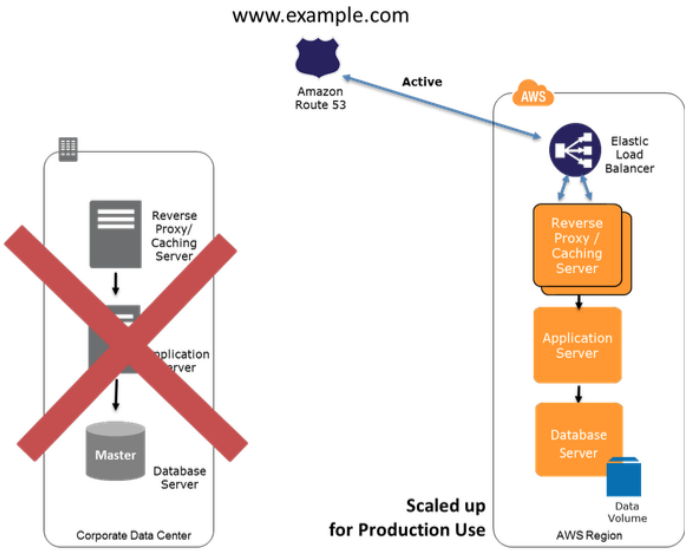


Figura 4.14: Fase de Recuperação do *Multi-site Solution*
Referência: [11]

Diversas questões devem ser levadas em consideração quando se opta por construir um Data Center. Há a questão geográfica, de energia, de capacidade, projeto para passagem de cabos através de pisos elevados, segurança física, segurança lógica, custos de manutenção com equipamentos, depreciação e outros. A estrutura e o esforço necessários para essa construção é muito grande.

Um Data Center organizado e bem estruturado vem acompanhado também de alto custo. A figura 4.15 ilustra parte da sala de um dos Data Centers do Facebook, localizado na Suécia. Através dela é possível verificar a grande dimensão do espaço utlizado.

Figura 4.15: Racks de um Data Center do Facebook, localizado na Suécia
http://static.dpr.com/assets/project-media/facebook-sweden-datacenter-racks2.jpg

O projeto de um novo Data Center para o Facebook é ilustrado na figura 4.16

Figura 4.16: Projeto de Datacenter do Facebook localizado na Carolina do Norte
http://i1-news.softpedia-static.com/images/news2/Facebook-to-Build-a-Second-450-Million-Data-Center-2.jpg

Manter a replicação de um Data Center de grande porte em outra localização para utilização de *Disaster Recovery* é um custo que é apenas compatível com empresas que possuem grande receita, não sendo acessíveis aos pequenos empresários e a grande parte das *startups*.

De acordo com o livro de Manoel Veras [20], segundo a IBM, 85% da capacidade de computação do mundo está ociosa. Esse dado mostra que manter um Data Center que é projetado de forma a atender o crescimento da empresa por alguns anos pode significar em grande disperdício de dinheiro quando comparado ao modelo de *Cloud Computing*, onde os recursos se adaptam na medida do necessário no modelo *pay-as-you-go* ilustrado na figura. Se a alta disponibilidade é necessária e há um grande acréscimo de acessos de

forma sazonal, será necessário adquirir hardware suficiente para suprir essa demanda que é apenas sazonal, sendo grande parte desses recursos não utilizados fora do período de sazonalidade. A diferença entre o hardware comprado e o recurso de utilizado é traduzido em desperdício.

As figuras 4.17 e 4.18 ilustram a capacidade da *Cloud* Pública de escalabilidade sob demanda.

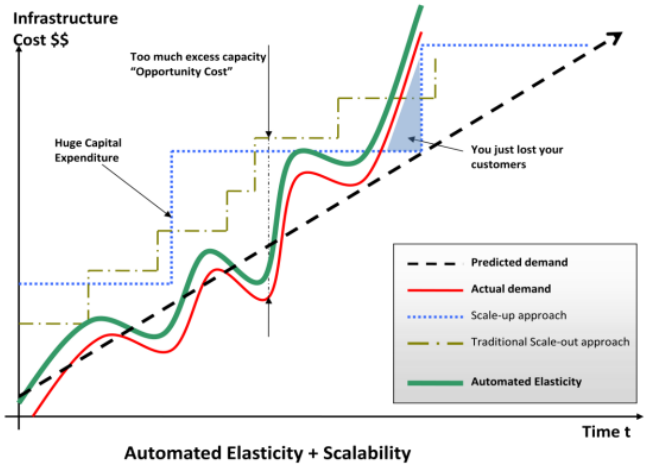


Figura 4.17: Gráfico - Custo versus Tempo
http://www.esds.co.in/blog/what-is-saas-cloud-truth

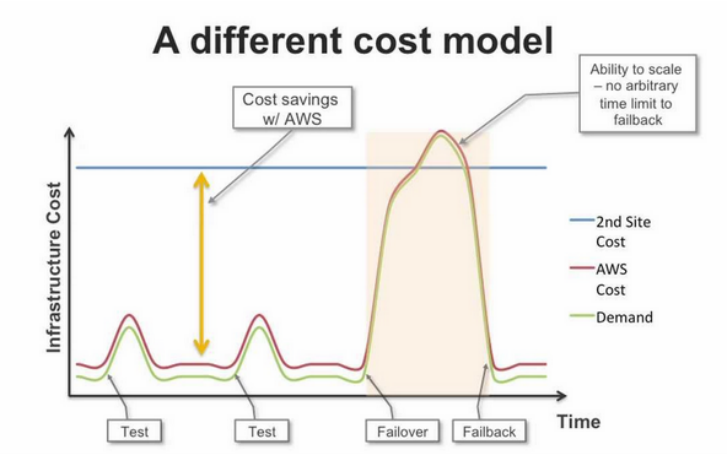


Figura 4.18: Gráfico2 - Custo versus Tempo
Referência: AWS Summit NYC 2012: Disaster Recovery with the AWS Cloud

Outro benefício da adoção do modelo de *Cloud Computing* é a redução dos custos de energia. Essa redução se deve ao fato de grandes provedores deste tipo de serviço escolherem localizações de menor custo energético. Sendo assim, o valor referente a energia que está incluído na taxa cobrada pelo provedor é certamente menor do que o valor energético gasto por máquina em um Data Center de pequeno porte localizado em grandes

cidades (onde há maior concentração de empresas).

Um artigo publicado por pesquisadores da *University of Massachusetts Amherst* e da *ATT Labs - Research* em 2010 [19], mostrou uma comparação de custo entre os modelos de *cloud* pública e *colocation* para uso de *Disaster Recovery*. A decisão por comparação entre esses dois modelos foi devido ao fato do custo de replicação em outro Data Center ser muito elevado, não sendo considerada opção para a maioria das empresas e instituições.

Para a comparação foram consideros os custos por ano para cada um dos modelos considerando um *uptime* de 99%. O custo foi comparado com base no uso de instâncias do serviço Amazon EC2 e o de colocation, cuja empresa não foi referenciada no artigo. Considerando esse tempo de *uptime*, foi calculado o custo de se manter uma determinada arquitetura por 3.6 dias nos dois ambientes.

Os resultados são ilustrados na figura 4.19:

	Public Cloud		Colocation	
RUBiS	Replication	Failover	Replication	Failover
Servers	$2.04	$32.64	$26.88	$26.88
Network	$0.54	$18.00	$1.16	$39.14
Storage	$1.22	$1.39	–	–
Total per day	$3.80	$52.03	$28.04	$66.01
Total per year	$1,386	$18,992	$10,234	$24,095
99% uptime cost	**$1,562 per year**		**$10,373 per year**	

(a)

Resource Consumption		
	Replication	Failover
Servers	1 cloud / 4 colo	4
Network	5.4 GB/day	180 GB/day
Storage	30 GB	30 GB
IO	130 req/sec	150 req/sec

(b)

Figure 2: (a) Cost per day and year for providing DR services for RUBiS. Under normal operation, only the Replication Mode cost must be paid, leading to substantial savings when using a cloud platform. (b) Resources required during Replication and Failover Modes are the same for the cloud and colocation center except that the colo center must always have 4 servers available.

Figura 4.19: Comparação entre o modelo de *Cloud* Pública e *Colocation* para Aplicação Web

Referência: [19]

Considerando que os desastres são raros, foi possível verificar uma redução de 85% do custo quando opta-se pelo modelo de DR em *Cloud* Pública.

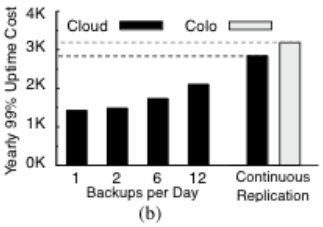
Outro estudo de caso deste artigo é referente ao custo de prover DR para uma aplicação *Data Warehouse*. Nesse caso foi considerado um *Data Warehouse* pequeno, com apenas 1TB de capacidade e com adição de 1GB de dados por dia utilizando custos baseados na instância *"High-Memory Extra Large Instance*

do Amazon EC2. Nesse caso, também considerando *uptime* de 99%, os resultados mostraram que o custo total por ano também foi menor para o modelo de *Cloud* Publica. Os resultados são exibidos na figura 4.20

Conclui-se então que o modelo de *Disaster Recovery* utilizando *Cloud* Pública

	Public Cloud		Colocation	
Data Warehouse	Replication	Failover	Replication	Failover
Servers	$4.08	$12.00	$8.51	$8.51
Network	$0.10	$0.12	$0.22	$0.26
Storage	$3.50	$3.92	–	–
Total per day	$7.68	$16.04	$8.73	$8.77
Total per year	$2,802	$5,853	$3,186	$3,202
99% uptime cost	**$2,832 per year**		**$3,186 per year**	

(a)



(b)

Figure 3: (a) Cost for providing DR services for the data warehouse application. The cloud provides only moderate savings due to high storage costs. (b) Using periodic backups can significantly lower the price of DR in the cloud by reducing the cost of VMs.

Figura 4.20: Comparação entre o modelo de *Cloud* Publica e *Colocation* para *Data Warehouse*

Referência: [19]

Capítulo 5

Monitoramento de Sistemas

O pior cenário para a prestação de um serviço de qualidade é aquele em que o cliente ou usuário informa ao prestador que o serviço não está funcionando ou está degradado, e o prestador não sabe previamente o motivo da indisponibilidade ou degradação. Por isso o monitoramento é de extrema importãncia para o garantir a qualidade do serviço e manter uma boa imagem perante seus usuários ou clientes finais.

5.1 A importância de um ambiente monitorado

Recente pesquisa realizada pela Opinion Matters a pedido da Monitis [15] consultou americanos que realizam compras *online* e constatou que 56% dos consumidores que gastam mais de duas horas por semana em compras cancelam uma ou mais operações devido a um erro na página ou um tempo de resposta muito lento e 86% afirmaram que abandonariam um serviço já contratado se encontrassem um similar porém mais rápido. Além disso, 74% de todos os usuários de serviços *online* entrevistados informaram que mudariam de prestador caso obtivessem uma experiência de uso melhor que a do concorrente.

Nos tempos atuais somente o monitoramento da saúde e desempenho dos recusos físicos da rede não é suficiente para garantiar a qualidade dos serviços. É preciso se aproximar mais da experiência do usuário, é importante ter mecanismos e agentes que testará todas as camadas do serviço. Desde a saúde dos roteadores de borda até o tempo de resposta a uma requisição. Atualmente há diversas soluções SaaS de monitoração no qual não é necessário uma estrutura física para monitorar o ambiente e os seus serviços. Inclusive estas soluções são projetadas para serem escaláveis e flexíveis, permitindo aos usuários adicionar monitoramentos conforme a sua necessidade.

Visto isto, foi incluído na proposta desenvolvida um modelo de monitoramento através de um dos líderes de mercado em inovação [10] no monitoramento de desempenho de aplicações chamado New Relic.

Figura 5.1: Compatativo entre os *players* de monitoramento de desempenho de aplicação
Referência: [10]

5.2 New Relic

O New Relic é um dos muitos desenvolvedores de soluções para monitoração do desempenho de aplicações existentes no mercado. Porém o seu diferencial está na fácil integração com o código da aplicação. Ao contrário outras soluções, o New Relic não se apresenta de forma intrusiva nesta integração tornando obrigatória a alteração significativas do código fonte da aplicação. Pelo contrário, o New Relic possui bibliotecas prontas cabendo ao desenvolvedor somente a inclusão destas em seu código. Esta solução, chamada de *Application Performace Monitor* (APM) permitiu, por exemplo, a empresa *Global Technology Manufacturer* uma economia de US$3 milhões [18] com monitoração dos seus serviços. Este tipo de abordagem na forma de monitoração permite a empresa maior agilidade na coleta dos dados de desempenho mesmo quando novas versões são inseridas em produção

uma vez que a integração é definitiva.

Esta solução é paga para monitoração do desempenho de aplicações, porém para monitorar recursos como utilização de processadores ou tráfego de rede o New Relic disponibiliza agentes gratuitos para serem instalados gerando painéis como o da Figura 5.2

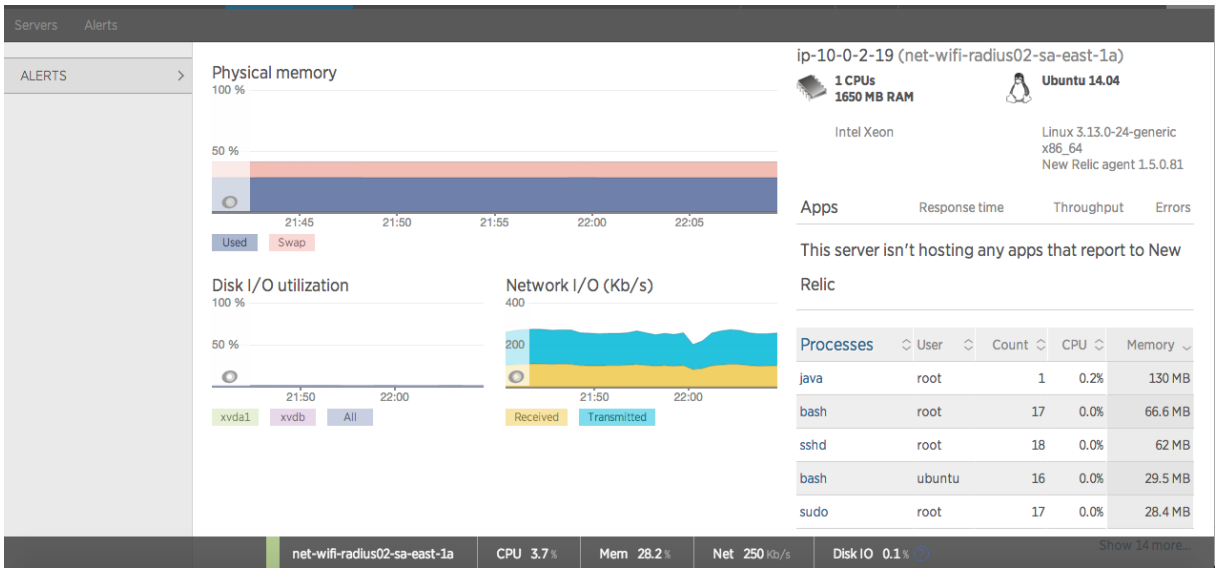


Figura 5.2: Painel de monitoração de serviços como tráfego de rede e memória física

No painel disponibilizado para a aplicação é possível visualizar o status de todas as aplicações configuradas conforme Figura 5.3. O usuário pode ordenar conforme a métrica que desejar sendo recomendado a ordenação automática conforme nível de criticidade.

Telas com gráficos do desempenho da aplicação pode ser vistas com um histórico que pode ser de 30 minutos ou até 3 meses. Na Figura 5.4 pode ser visto os gráficos da aplicação Api Graduação que é responsável pela exibição de dados de alunos da Universidade Federal Fluminense. Esta solução também disponibiliza mapas de relacionamentos e dependências entre as aplicações de forma automática conforme pode ser visto na Figura 5.6 para uma aplicação responsável pela inserção de créditos em dispositivos móveis.

Além de parâmetros como tempo de resposta e *throughput* através do New Relic é possível verificar o índice de qualidade da aplicação através do *Application Performance Index*, um método de avaliação da desempenho de aplicações calculada através do nível desempenho tolerável dentro de uma amostra. Na Figura 5.5 é exibida a Apdex de uma aplicação responsável pela autenticação de usuários em uma rede *Wireless outdoor*.

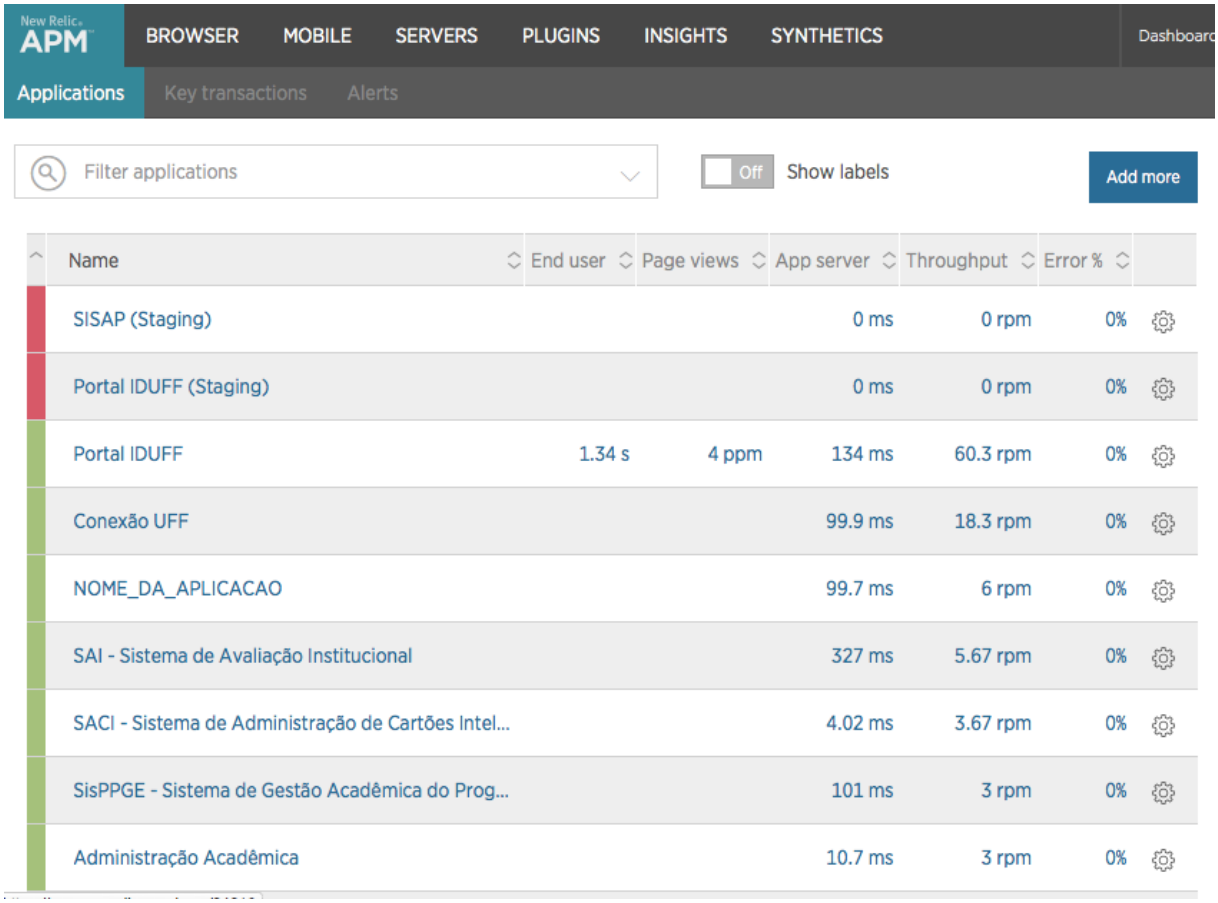


Figura 5.3: Painel de monitoração das aplicações

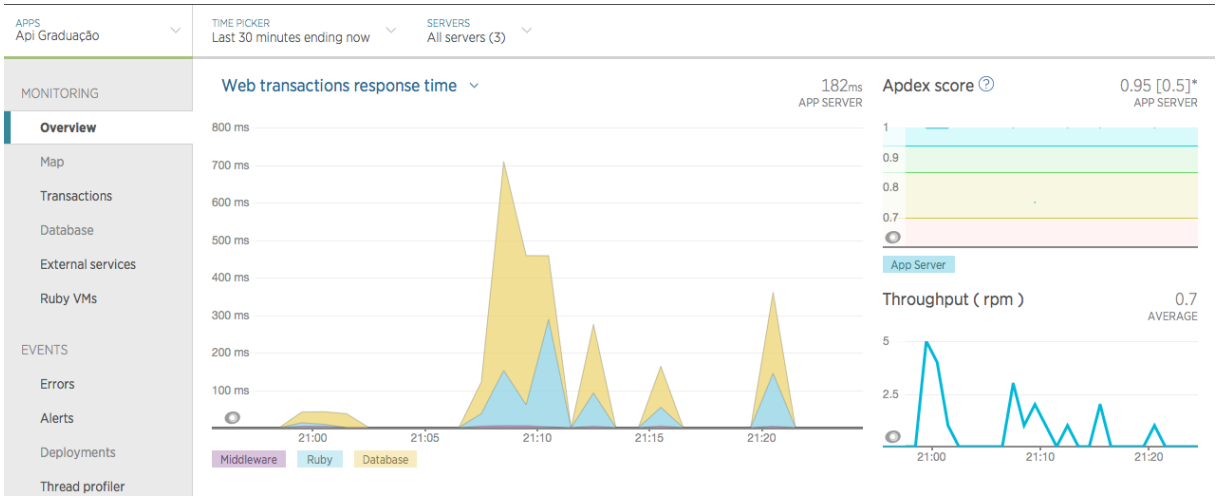


Figura 5.4: Gráficos de desempenho da aplicação Api Graduação

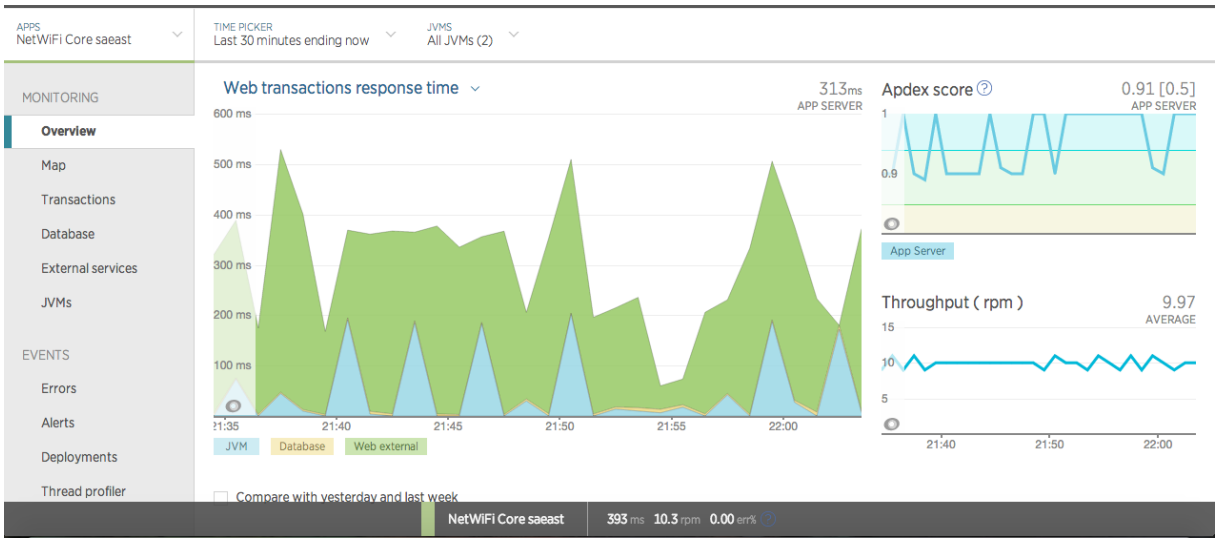


Figura 5.5: Gráfico de Apdex no canto superior direito.

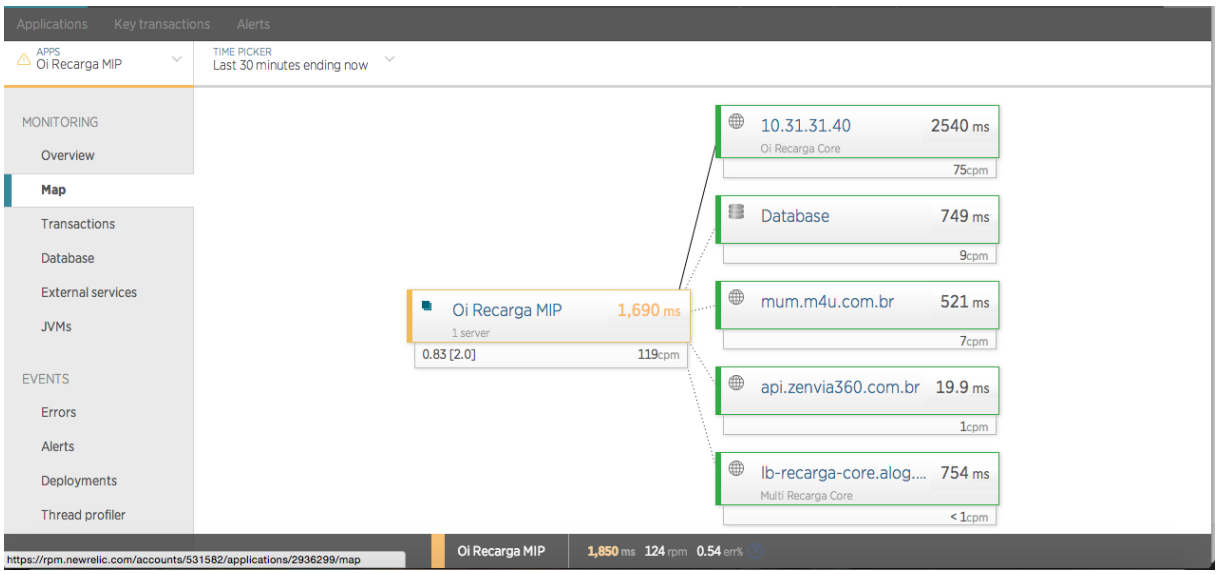


Figura 5.6: Mapa de relacionamento

Capítulo 6

Proposta de Solução

6.1 Introdução

Como proposta de solução para o problema de downtime de grande parte das empresas, foi desenvolvido um software que tem como objetivo ajudar na recuperação de desastres de forma simples e rápida. Para isso desenvolvemos uma solução que integra a grande eficiência e simplicidade do monitoramento utilizando o NewRelic com a grande praticidade de se colocar uma aplicação no ar utilizando o serviço Elastic Beanstalk da Amazon AWS.

Outro componente importante da solução, que por enquanto é obrigatório, mas que poderá deixar de ser é o uso de containers Docker para criar um ambiente virtual em que a aplicação é inserida. Esse conceito será melhor explicado na sessão correspondente a ele, mas não será aprofundado visto que é apenas um requisito para o funcionamento da aplicação, mas que foge do foco do trabalho.

6.2 Tecnologias utilizadas

6.2.1 Ruby on Rails

O Ruby on Rails é um framework para desenvolvimento web opensource que vem sido amplamente utilizado nos últimos anos, ganhando destaque pela sua simplicidade e rapidez na que é possível desenvolver features. Algumas grandes instituições, como a Universidade Federal Fluminense, já utilizam esse framework em grande parte de suas aplicações. Com a adoção do mesmo, foi possível um ganho de produtividade e confiabilidade muito grande que resultou em mais sistemas criados em menos tempo, gerando menor custo. A escolha

pelo framework foi devido a essas facilidades e também pelo maior domínio já existente sobre a ferramenta. Exemplo de empresas que utilizam Ruby on Rails em produção são: Groupon, ESPN, NewRelic, Github, Shopify, Bloomberg, Airbnb, SoundCloud e outros.

Podemos enxergar o framework Rails como uma ferramenta para desenvolvimento web utilizando a linguagem de programação Ruby, assim como o framework web Django é utlizado para a linguagem Python ou o Spring MVC para o Java.

Aplicações desenvolvidas em Ruby on Rails são divididas em camadas que são conhecidas como MVC – Model, View e Controller.

Um esquema que podemos utilizar para ilustrar essa arquitetura é representado abaixo:

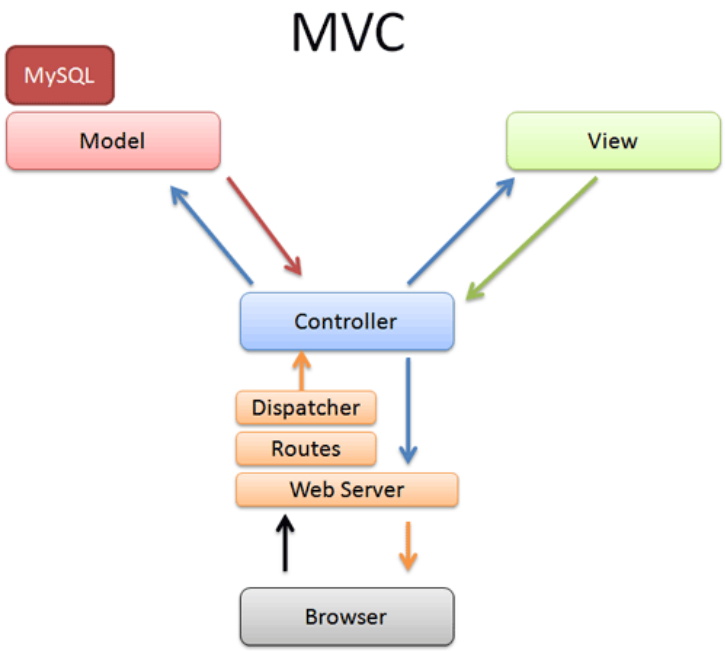


Figura 6.1: Ruby on Rails - Arquitetura MVC

Quando a requisição é feita pelo browser, ela passa pela camada de Web Server, onde podem ser usados as mais diversas ferramentas, porém sendo comumente utilizado em produção o Apache ou o Nginx. Quando a camada de Web Server recebe a requisição, ela encaminha para o módulo do Rails responsável pelo roteamento (*ActionDispatch::Routing*) que verifica qual endereço está sendo requisitado, encaminhando então para o *controller* correspondente de acordo com um arquivo pré-definido (routes.rb) que contém todas as rotas cadastradas pelo desenvolvedor.

A camada de *controller* é responsável por receber os parâmetros da requisição que foram encaminhados pelo *routes* e produzir o output apropriado. Para isso, muitas vezes

é necessária a comunicação com as outras camadas: *Model* e *View*. A camada de modelo contém regras de negócio, regras de consistência em banco de dados (como por exemplo, quais colunas são obrigatórias) e é responsável por interagir diretamente com o banco de dados, tendo a capacidade de salvar, apagar, buscar ou inserir dados no mesmo.

A camada de View é responsável por apresentar ao usuário a saída que é vista comumente em HTML. Ela também tem a capacidade de se comunicar com outras camadas, como o *model* e o *controller*. Isso acontece quando queremos exibir na tela informações contidas em banco de dados, como por exemplo o nome e matrícula dos alunos de Engenharia de Telecomunicações UFF, ou até mesmo mensagens personalizadas indicando o sucesso de uma determinada ação. O ideal é que não seja inserida lógica de programação diretamente na View para que tenhamos baixo acoplamento, conceito visto em Engenharia de Software.

Como resumo da arquitetura em funcionamento, o cenário de um aluno efetuando seu login no sistema acadêmico da UFF e como essa requisição interage em meio as camadas do MVC é apresentado abaixo.

Na ordem, o que ocorre é:

1. O aluno acessa um endereço, como por exemplo https://sistemas.uff.br/iduff
2. O DNS é resolvido para um IP, que contém um Web Server sendo executado em determinada porta
3. A requisição é direcionada para esse Web Server
4. O Web Server direcionada a requisição para o ActionDispatch do Rails
5. O ActionDispatch compara o endereço pedido com as rotas previamente escritas no arquivo routes.rb e envia para o Controller responsável por receber aquela requisição
6. Caso a requisição tenha sido feita para uma página de index, o controller irá se comunicar com a View e solicitar que a página seja renderizada
7. A requisição volta para o browser do aluno que agora pode navegar normalmente

Caso o aluno efetue login, por exemplo, o *controller* também fará comunicação com o *model* para que os campos preenchidos com CPF e senha possam ser validados no banco de dados. Sendo a operação efetuada com sucesso ou não, a *view* deverá ser capaz de retornar uma mensagem como “Login efetuado com sucesso” ou “CPF e/ou senha inválidos”.

Essa arquitetura apresentada acima foi utilizada em todo o software, que será explicado com mais detalhes nas seções adiante.

6.2.2 WebServices

WebServices são comumente utilizados para que seja possível a interoperabilidade entre sistemas. Isso quer dizer que através do uso deste, é possível que duas ou mais aplicações se comuniquem e troquem informações através do protocolo HTTP. Uma grande vantagem é que a troca de mensagens entre as aplicações é feita através de um padrão geralmente apresentado nos formatos XML, JSON ou SOAP. Sendo assim, é possível que aplicações escritas em linguagens distintas consigam se comunicar.

Quando temos um ambiente onde as aplicações se comunicam através dos WebServices, ganhamos a liberdade de resolver problemas computacionais usando a linguagem que melhor convir para a solução do mesmo. Esse conceito foi fortemente empregado no software proposto neste trabalho.

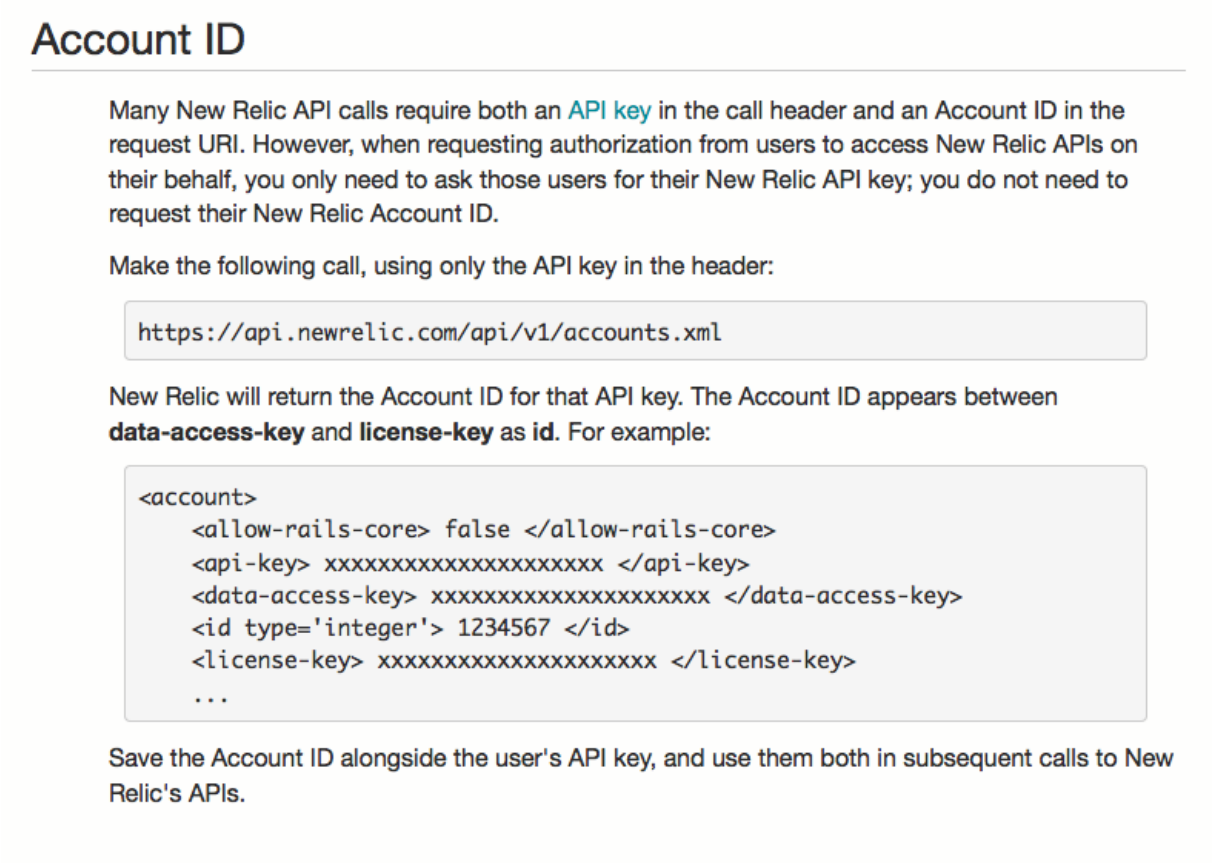
Imagine se fosse necessário consultar um determinado atributo de uma pessoa (data de nascimento), que apenas outra aplicação tem acesso direto (via banco de dados), e então fosse necessário liberar acesso para a cosulta desse atributo em determinada tabela do banco de dados para um determinado IP (onde provavelmente a aplicação que deseja consumir o recurso está hospedada) apenas para que essa consulta possa ocorrer.

Se a aplicação precisa de uma informação que está contida de forma espalhada em diversos bancos de dados diferentes esse cenário é ainda mais crítico. Imagine ainda um cenário em que a UFF necessita de acesso a dados que apenas o Ministério da Educação é capaz de fornecer em tempo real. Para que isso seja possível, sem o uso de WebServices, você deveria solicitar acesso a todos esses bancos de dados. Temos assim um cenário de caos generalizado, impossível de manter e garantir segurança a médio/longo prazo.

Para a utilização de um WebService, é necessário que o responsável pela implementação deste crie uma documentação confiável que represente de forma fiel o retorno dos dados para uma certa entrada. É comum que seja exigido, caso o serviço não seja público, uma chave de identificação do requisitante para que o serviço seja atendido. Um detalhe importante é que, caso as informações trafegadas sejam privadas ou contenham dados sensíveis, deve-se utilizar o protocolo SSL para que as requisições não sejam interceptadas e capturadas em texto plano por ataques do tipo *man-in-the-middle*.

Um exemplo de documentação que foi usada neste projeto para comunicação com a

ferramenta de monitorameto NewRelic está ilustrada abaixo:



Account ID

Many New Relic API calls require both an API key in the call header and an Account ID in the request URI. However, when requesting authorization from users to access New Relic APIs on their behalf, you only need to ask those users for their New Relic API key; you do not need to request their New Relic Account ID.

Make the following call, using only the API key in the header:

```
https://api.newrelic.com/api/v1/accounts.xml
```

New Relic will return the Account ID for that API key. The Account ID appears between **data-access-key** and **license-key** as **id**. For example:

```
<account>
    <allow-rails-core> false </allow-rails-core>
    <api-key> xxxxxxxxxxxxxxxxxxxx </api-key>
    <data-access-key> xxxxxxxxxxxxxxxxxxxx </data-access-key>
    <id type='integer'> 1234567 </id>
    <license-key> xxxxxxxxxxxxxxxxxxxx </license-key>
    ...
```

Save the Account ID alongside the user's API key, and use them both in subsequent calls to New Relic's APIs.

Figura 6.2: Documentação de WebService do NewRelic
Referência: [17]

Na documentação acima é possível perceber que existe um atributo obrigatório chamado *API key* no header do HTTP. O endereço para o qual deve ser feita a requisição também é demonstrado. Comumente, apesar de não apresentado na documentação acima, é informado se a requisição deve ser feita utilizando GET ou POST. Por padrão, requisições que tem com objetivo leitura de dados são feitas utilizando GET, enquanto requisições que são feitas para gravação de novos dados ou atualização dos mesmos são feitas via POST.

Também é exibido a forma como o WebService retornará a sua requisição. Essa informação é muito importante para que seja possível efetuar o parse correto dos dados retornados.

Note que esse WebService retorna, para uma determinada *API key*, o ID da conta do usuário que está vinculado a essa API. É exibido também o formato e a ordem em que as informações são retornadas. Caso seja necessário consumir esse serviço e gravar em um banco de dados local apenas o data-access-key, seria necessário fazer o parse dessa tag

específica do XML de retorno para então salvar este valor.

Espera-se também, como boa prática, que os devidos status HTTP sejam retornados corretamente. Caso os dados sejam retornados com sucesso, o status retornado deverá ser 200 OK, caso ocorra alguma falha de autenticação, o retorno esperado recebe o código 401, caso o ítem que você esteja buscando (geralmente enviado como parâmetro) não seja encontrado, o retorno deve conter o código 404, e assim por diante. Esses códigos, quando utilizados de forma correta ajudam o desenvolvedor a tratar possíveis problemas sem depender de implementações do servidor, como por exemplo a comparação de uma determinada palavra, que causaria grande acoplamento, implicando em grande risco de falha na integração caso essa palavra seja alterada pelo desenvolvedor do WebService.

6.2.3 Arquitetura da Solução

Conforme visto na seção anterior, o projeto funciona através do uso de WebServices para comunicação com as plataformas de *Cloud Computing* da Amazon AWS e monitoramento, do NewRelic. A grande vantagem de se adotar arquiteturas como essa é a maior capacidade que o sistema adquire de adequar a diversas alternativas de *Cloud Computing* e monitoramento, visto que essas duas não são as únicas opções disponíveis no mercado. Sendo assim, futuramente será possível integrar com plataformas que também tenham suporte a comunicação via *WebServices*, aumentando as alternativas para o cliente.

O projeto é composto pelos seguintes módulos:

1. Aplicação principal, acessível pelo cliente, desenvolvida utilizando o framework Ruby on Rails
2. Banco de Dados MySQL
3. Módulo desenvolvido em Python, utilizando a biblioteca web.py, responsável pela comunicação entre a aplicação principal e os WebServices da Amazon AWS

A comunicação com o NewRelic é feita através da própria aplicação principal, diferindo nesse ponto da arquitetura utilizada para comunicação com a Amazon AWS. A imagem abaixo ilustra os componentes da solução.

A diferença na forma de implementação da comunicação entre Rails WebServer - Amazon AWS e Rails WebServer - NewRelic foi dada devido a dificuldade de utilizar o SDK Ruby [3], provido pela Amazon, para comunicação com o módulo responsável por

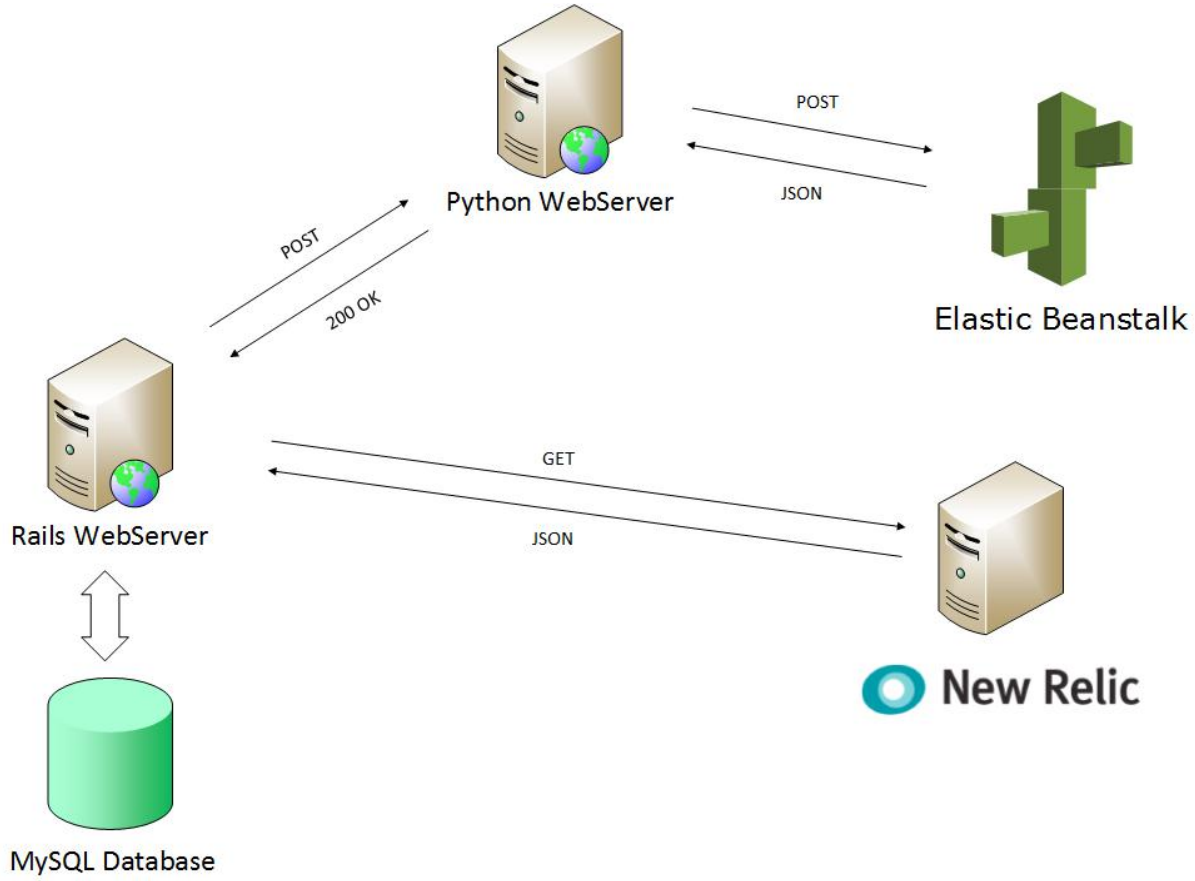


Figura 6.3: Arquitetura utilizada no projeto

fazer o deploy do sistema que será recuperado. Diante desse problema, essa comunicação foi implementada utilizando o SDK Python [2], sendo então a escolha pela linguagem Python baseada neste fato.

Através da análise da imagem fica claro entender como o sistema funciona e suas dependências. O *Web Server* que ficará hospedado na infra-estrutura que o cliente desejar, faz comunicação com o NewRelic para que seja possível exibir os sistemas que estão sendo monitorados e, caso um sistema esteja *offline*, é possível efetuar o *deploy* do mesmo através da requisição feita ao Python WebServer, que por sua vez se comunica com o serviço Elastic Beanstalk da Amazon, faz o *parse* do JSON recebido como resposta e devolve um *Status HTTP* para o Rails WebServer informando se o início do deploy ocorreu com sucesso.

6.2.4 A modelagem do Banco de Dados

O Banco de Dados foi modelado inicialmente de forma a contemplar um ambiente de teste, sendo utilizado como prova de conceito. Para isso, o modelo ainda se encontra

bastante simplificado e pode crescer de acordo com o crescimento dos requisitos planejados para próximas versões da aplicação.

O diagrama de Entidade Relacionamento pode ser visto abaixo.

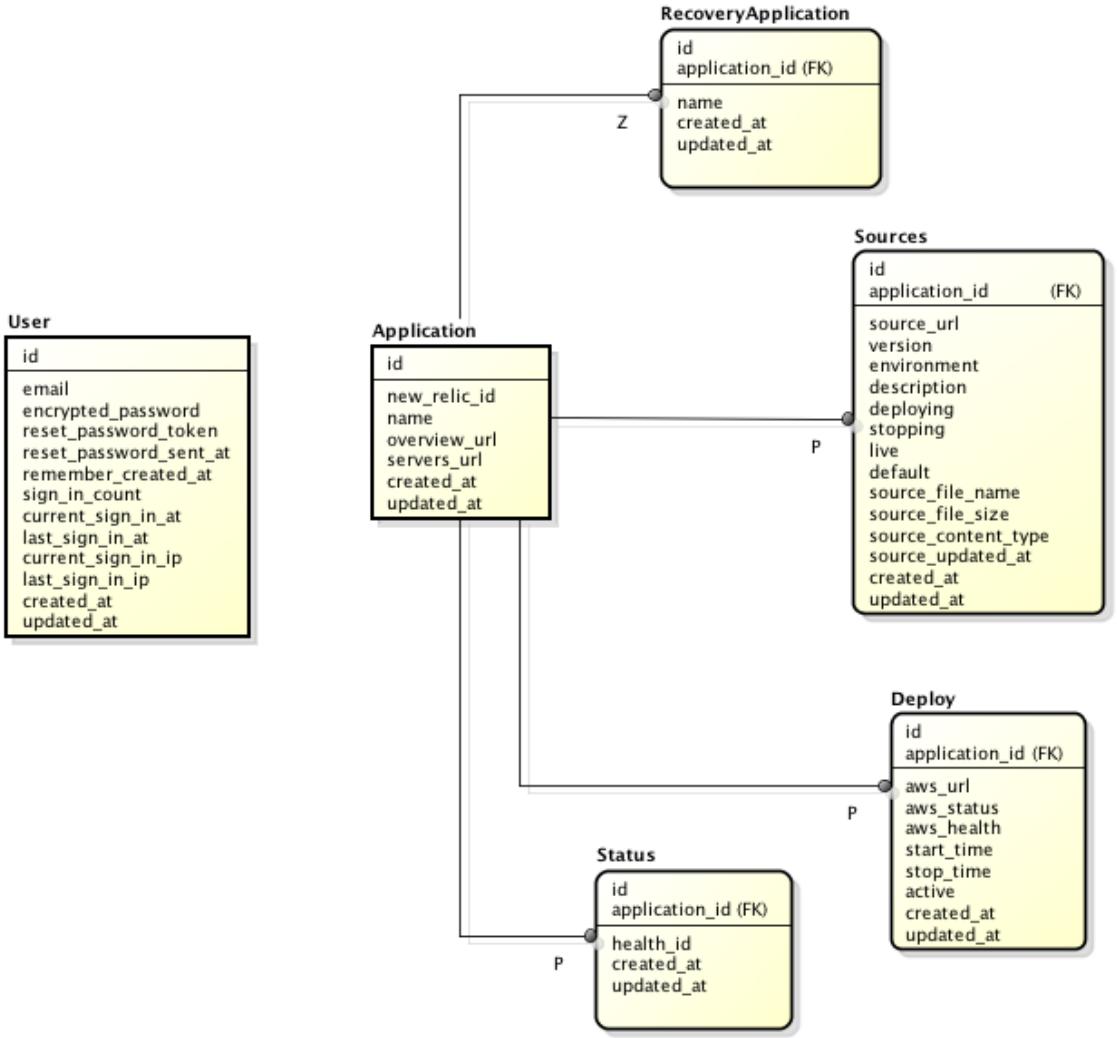


Figura 6.4: Diagrama Entidade Relacionamento

Através dele podemos perceber que a principal tabela é a *Application*, pois tem relacionamento com quase todas as demais. Os dados contendo essas aplicações são inseridos de acordo da sincronização com o NewRelic. Sendo assim, não há trabalho prévio para o utilizador da solução no que diz respeito ao cadastro de cada uma delas. Esse trabalho de cadastro das aplicações seria árduo caso o utilizador administre uma quantidade grande de aplicações.

A tabela *RecoveryApplication* se fez necessária para que fosse possível realizar o acompanhamento correto da aplicação que está sendo recuperada. Essa necessidade é justificada com mais detalhes na próxima seção.

A tabela *Sources* armazena informações como o nome do código fonte relacionado a aplicação e dados sobre o *deploy* corrente relacionado ao mesmo.

Quando o *deploy* da aplicação é feito na Amazon, a tabela *Deploy* inicia seu preenchimento para que seja possível acompanhar o *status* deste, o momento em que foi iniciado e o momento em que a aplicação foi retirada do ambiente de *Cloud Computing*.

A tabela *Status* contém um acompanhamento sobre o monitoramento da aplicação feito pelo NewRelic. Nela são armazenados, a cada sincronismo, se a aplicação está *online*, *offline* ou instável.

6.2.5 Replicação dos dados do cliente

Quando tratamos de replicação de sistemas é comum que o assunto relacionado a replicação dos dados seja discutido. Nos dias de hoje, quase qualquer sistema lida com dados, e estes são de fundamental importância para a sobrevivência de grande parte das empresas. O valor de mercado de muitas está diretamente relacionado com a quantidade e com a qualidade dos dados que ela contém.

Tendo em vista esse cenário crítico, uma atenção especial se faz necessária a esse tema. Ao lidarmos com *Disaster Recovery*, deve-se pensar em todo o ambiente contido no DataCenter, porém o foco deste trabalho é específico no que diz respeito a aplicações web. Como o *DR* nem sempre é possível sem os dados disponíveis em tempo real, uma solução que deve ser abordada é a Replicação entre Bancos de Dados contidos no DataCenter e em ambiente de *Cloud Computing*.

Essa replicação deve ser realizada utilizando ferramentas disponibilizadas pela própria fabricante do seu Banco de Dados, que é quem tem total domínio sobre o funcionamento do mesmo e pode garantir maior integridade aos dados.

Grandes fabricantes de Bancos de Dados como Oracle, Microsoft e IBM, contém soluções já integradas que podem ser facilmente encontradas na internet ou em livros. Para isso é importante que se tenha um investimento no profissional que atua como *DBA (Database Administrator)* na empresa do cliente.

Replicações de dados são comuns em duas categorias:

1. Master-Master
2. Master-Slave

Bancos de dados gratuitos como o MySQL também implementam essa tecnologia, sendo então acessível à maioria. A principal diferença entre elas é que em replicações Master-Master, temos dos nós que são capazes de receber operações de escrita, enquanto no modelo Master-Slave, apenas o nó Master recebe escritas, sendo o Slave apenas para leitura.

A decisão relativa a qual dessas opções utilizar deve estar diretamente ligada a regra de negócio. Esse modelo funciona muito bem quando os requisitos são melhor distribuição de carga (*Load Balancer*) e redundância.

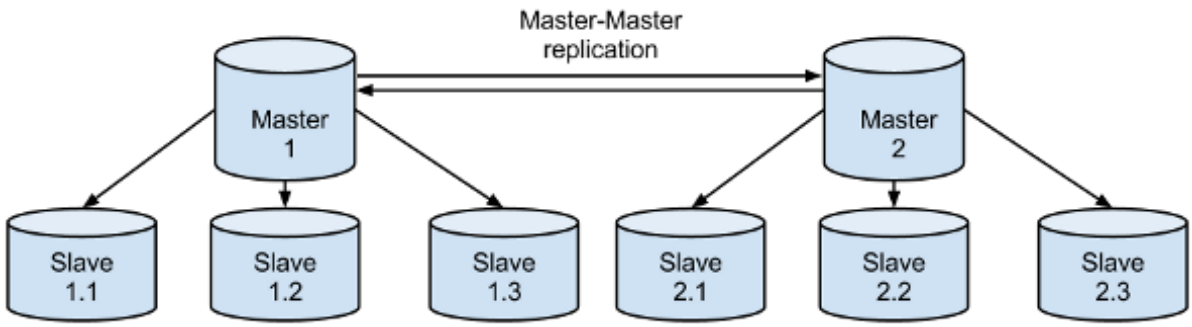


Figura 6.5: O modelo de Replicação dos Dados
Referência: [1]

De acordo com a solução proposta neste trabalho, para uma maior velocidade no momento em que ocorrer o desastre, é importante manter um Banco de Dados replicado em ambiente de *Cloud Computing*. Dessa forma, o código da aplicação deve contemplar que, caso esteja neste ambiente de recuperação, a mesma deve efetuar suas operações de leitura e escrita neste Banco de Dados, e não no que está localizado no DataCenter físico.

Com base em algumas pesquisas recentes é possível notar uma grande preocupação com relação a segurança dos dados hospedados em ambiente de *Cloud Computing* [14]. Quanto mais sigilosos são os dados, maior a preocupação em mantê-los em ambiente mais controlado, como um DataCenter próprio. A possibilidade de roubo dos dados por empresas concorrentes podem levar a falência para algumas das concorrentes. Por mais que empresas como a Amazon apresentem inúmeras certificações relativas a segurança [4], a desconfiança começou a se tornar maior com as famosas revelações do Edward Snowden, ex-funcionário da NSA - National Security Agency [12].

Dessa forma o convencimento da replicação dos dados de uma empresa em outro ambiente pode se tornar difícil para algumas empresas. Caso isso ocorra é possível optar por não replicar o Banco de Dados, efetuando a comunicação do sistema recuperado diretamente ao DataCenter físico. Essa solução funciona quando há desastre relativo

apenas à camada de aplicação, havendo então um risco de *downtime* por tempo maior do que o esperado caso o desastre impacte em todo o ambiente físico da empresa ou então a mesma perca conexão com a internet. Outra desvantagem na adoção dessa solução é o grande aumento da latência entre o Banco de Dados e a Aplicação. Esse aumento de latência, caso a aplicação efetue muitas operações de leitura e/ou escrita, pode ser muito crítico.

6.2.6 O software

O software tem como objetivo monitorar as aplicações já configuradas no seu sistema de monitoramento NewRelic e, caso necessário e solicitado pelo usuário, efetuar o *deploy* da aplicação em Amazon.

Uma vez estando com a aplicação configurada (em seu Data Center ou até mesmo em *Cloud* Pública, após realizar o acesso e efetuar o login, duas operações básicas são necessárias.

1. Sincronização com o NewRelic
2. Sinconização com a Amazon AWS

O primeiro é importante pois não é preciso cadastrar as aplicações. Conforme o número de aplicações gerenciadas aumenta em uma empresa, é cada vez mais difícil manter a aplicação de DR atualizada. Sendo assim, as aplicações são cadastras automaticamente a cada sincronização feita com o NewRelic, bastando apertar no botão *NewRelic Sync* localizado na barra superior. Essa sincronização pode ser visualizada na figura 6.6.

Feito isso, precisamos atribuir o código fonte da aplicação a ela em formato .zip ou então .war para o caso de aplicações Java. Além do código, dois outros arquivos devem estar contidos dentro desse arquivo:

1. Dockerfile
2. Dockerrun.aws.json

O primeiro arquivo especifica o ambiente virtual no qual a aplicação será executada dentro de um container Docker. A documentação que explica o que são esses containers e como configurá-los está disponível em https://docs.docker.com/. Uma explicação mais detalhada sobre essa tecnologia não será abordada neste trabalho.

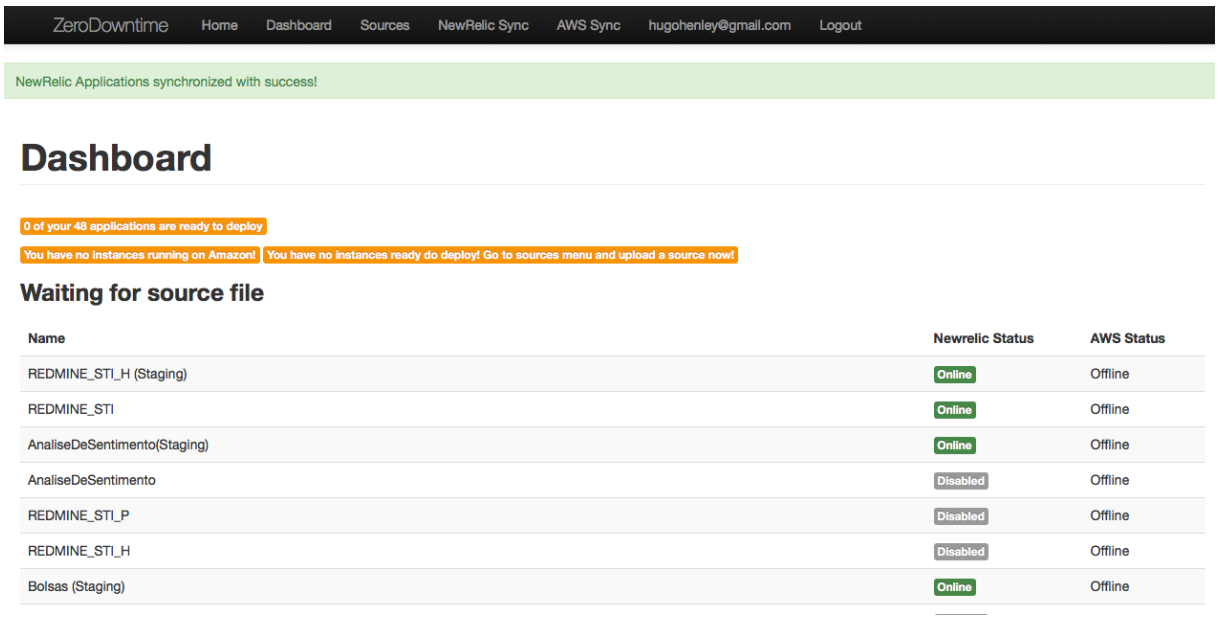


Figura 6.6: Sincronização das aplicações com o NewRelic

Um exemplo de Dockerfile pode ser visto na figura 6.7.

Dockerfile

```
FROM ubuntu:14.04

RUN apt-get update && apt-get upgrade -y

RUN apt-get install -y git git-core wget zip nodejs npm

EXPOSE 8080

# startup
ADD start.sh /tmp/
RUN chmod +x /tmp/start.sh
CMD ./tmp/start.sh
```

Figura 6.7: Dockerfile - Arquivo de configuração para criação de containers

O segundo arquivo contém informações como a versão do sistema operacional que será executado na sua aplicação e qual a porta deverá ser exposta nas regras de *firewall*. É através dele que o serviço de *deploy* da Amazon, Elastic Beanstalk, sabe como executar o container no qual a sua aplicação está contido.

Um exemplo deste arquivo pode ser visto na figura 6.8

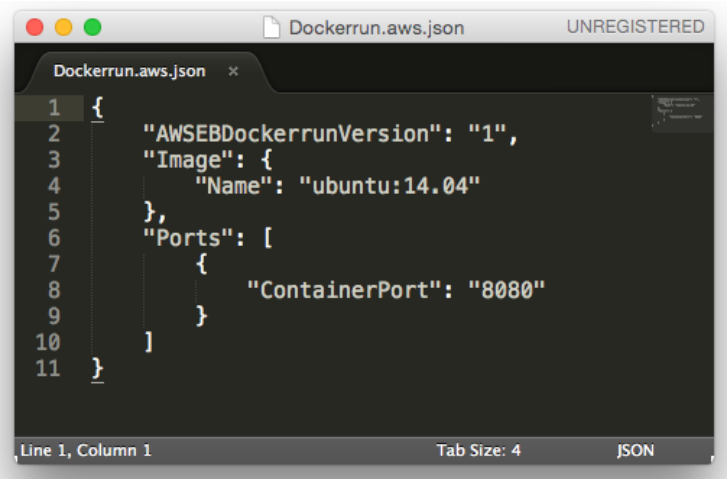


Figura 6.8: Dockerrun.aws.json

Após inserir estes arquivos no seu projeto e comprimir no formato zip, a vinculação de um código fonte a uma aplicação deve ser feita através do menu *Sources*, conforme pode ser visto na figura 6.9

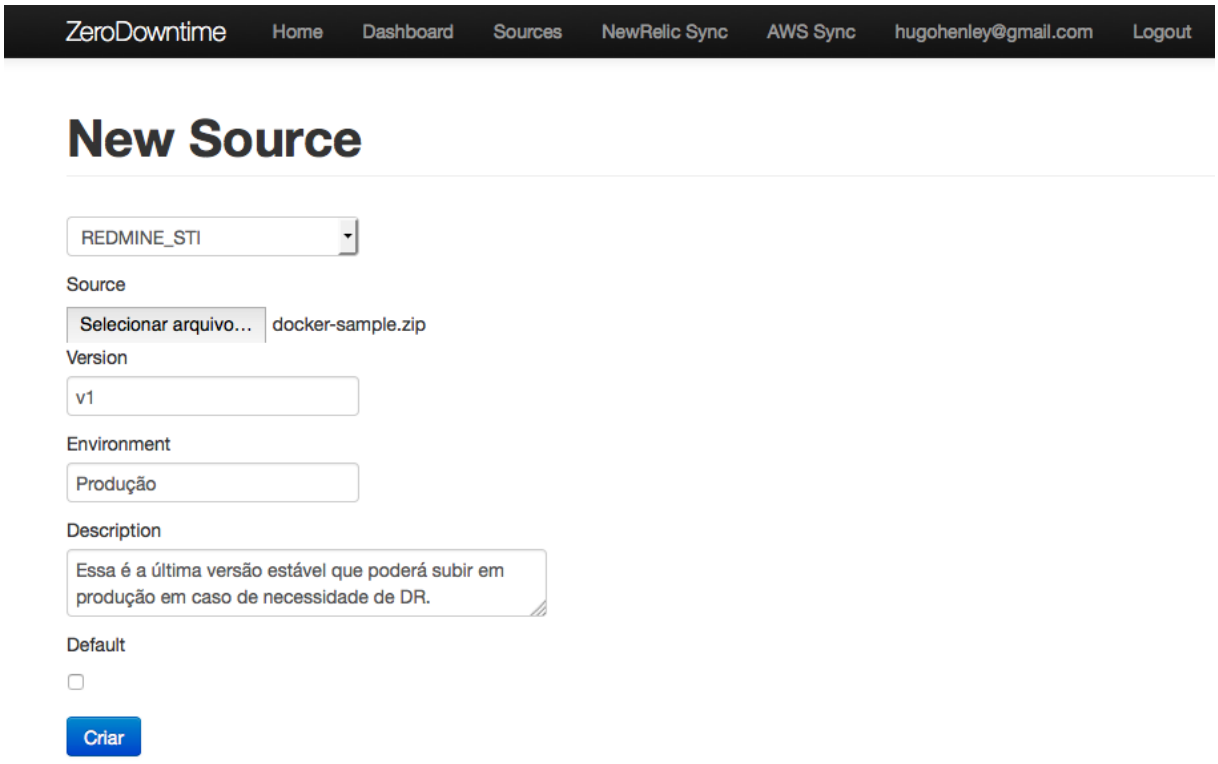
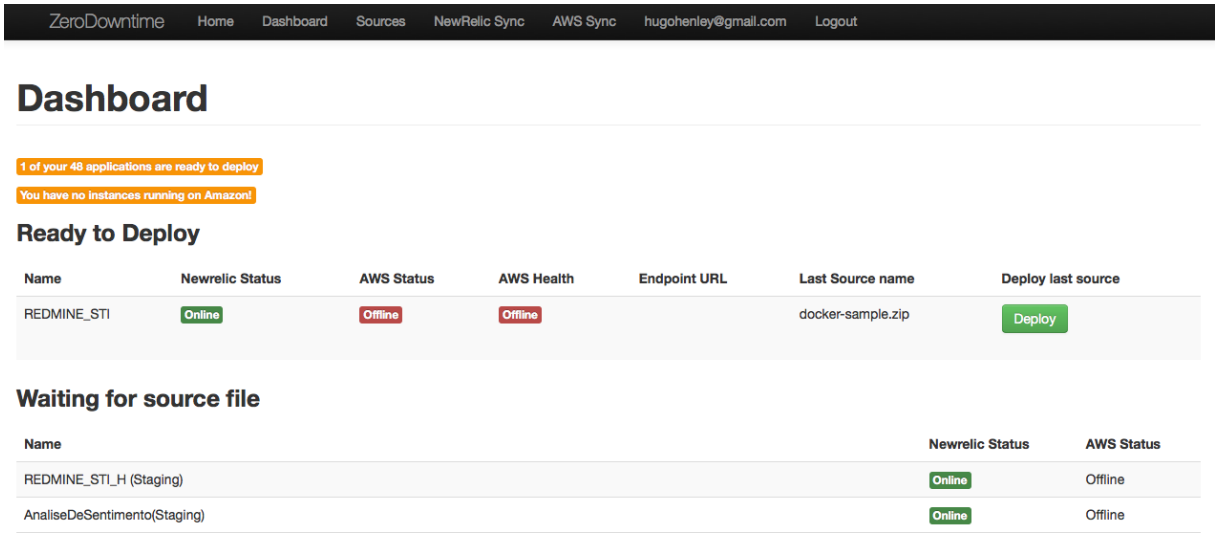


Figura 6.9: Vinculando um código fonte para a aplicação REDMINE STI

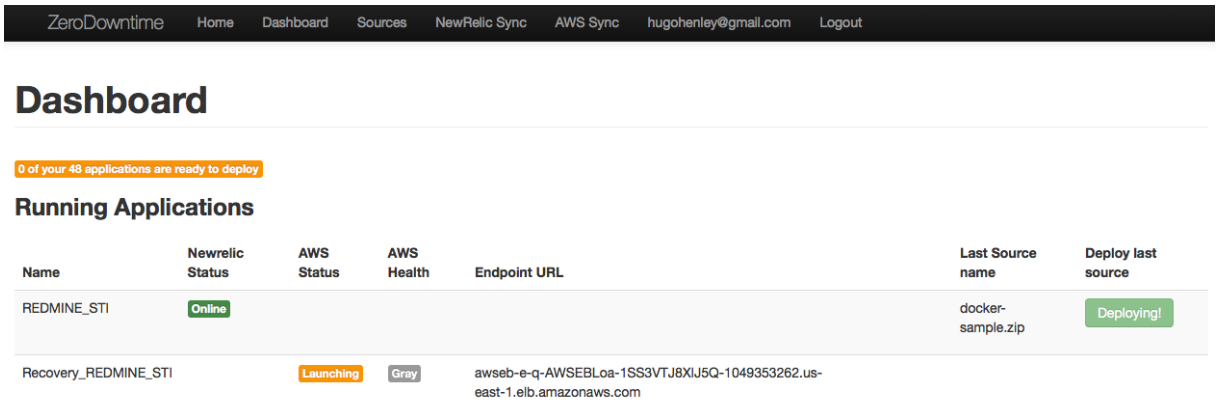
A partir desta vinculação já é possível efetuar o *deploy* da aplicação na Amazon, conforme ilustrado na figura 6.2.6

Ao clicar no botão Deploy, a aplicação enviará requisições para a API desenvolvida



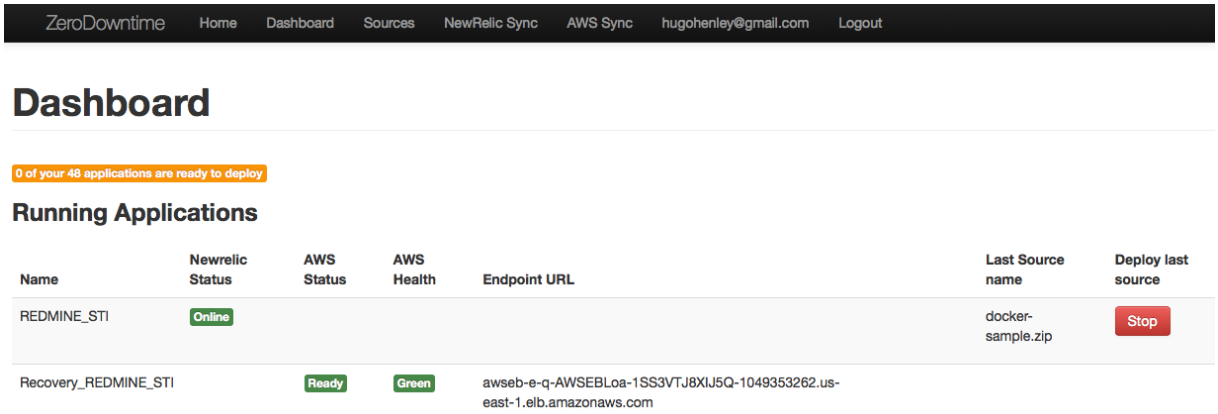
em Python para que ela inicie uma máquina virtual na Amazon e execute a aplicação. Nesse momento o dashboard exibirá algumas informações sobre a aplicação que estará sendo executada na Amazon. Através dele é possível acompanhar o momento em que a Amazon passa a disponibilizar a aplicação para acesso.

O momento em que o deploy está sendo executado e o momento em que a aplicação está disponível para acesso podem ser vistos nas figuras 6.2.6 e 6.2.6, respectivamente.



O endereço para acessar a nova aplicação está disponível em *Endpoint URL*. É para este endereço que o serviço de DNS deve ser alterado.

A partir deste momento é possível retirar a aplicação do ambiente da Amazon quando desejável.



6.2.7 Futuro

6.2.7.1 Novas funcionalidades

O objetivo com a criação desta aplicação foi criar uma prova de conceito para *Disaster Recovery* de aplicações *web*. Sendo assim, é possível apenas utilizar as funcionalidades básicas que foram suficientes para validar o conceito.

No futuro, é possível agregar ainda mais valor ao software desenvolvido, sendo alguns dos próximos objetivos de implementação os ítens:

1. Alteração automática do DNS no momento em que a aplicação passar a estar disponível em Amazon
2. Sincronização automática em um dado intervalo de tempo entre a aplicação e os serviços do NewRelic e da Amazon
3. Possibilidade de escolha de região da Amazon
4. Possibilidade de *deploy* de aplicações que não utilizam Docker
5. Análise preditiva que aponta uma grande probabilidade de, dado um determinado comportamento analisado através do NewRelic, uma aplicação sofrer instabilidades ou perda de conexão

Estes primeiros requisitos serão atendidos na próxima versão da aplicação, com exceção do último, que demanda um estudo maior mas que aumentará muito a proposta de valor da solução. Com a implementação deste último, a aplicação será capaz de provisionar o ambiente em *Cloud* pública de forma antecipada. Sendo assim, os usuários poderão

obter uma grande economia continuando com o uso do modelo Pilot-Light, mas obtendo um RTO ainda menor e, em alguns casos, próximo de zero.

6.2.7.2 Mobile

É díficil pensar nos dias de hoje sem o uso de aplicativos em *smartphones*. Essa realidade se faz cada vez mais presente, estes aplicativos são parte do nosso dia-a-dia. Uma proposta de aplicativo móvel se faz então necessária para que se possa navegar com facilidade e rapidez pelos recursos disponíveis pela aplicação de qualquer local.

Com a criação do aplicativo para dispositivos móveis que ocupam grande *market-share*, como Android e iPhone, será possível efetuar a recuperação de aplicações, monitoramento destas, exibir relatórios de uso, utilizar de recursos como *Push Notification* e outros.

Um protótipo do aplicativo mobile pode ser visto na figura 6.10

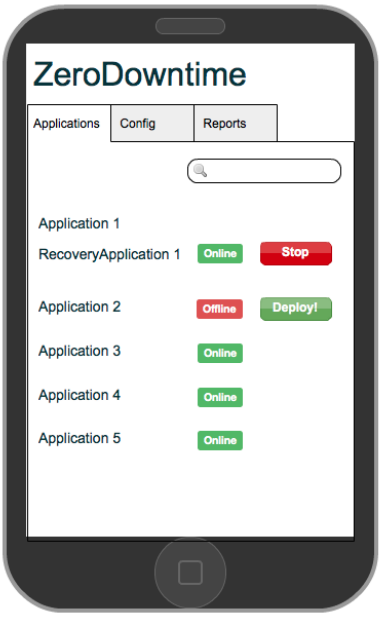


Figura 6.10: Protótipo de projeto mobile para o ZeroDowntime

Capítulo 7

Conclusão

De acordo com testes realizados durante a prova de conceito da aplicação de *Disaster Recovery* proposta, foi possível obter resultados satisfatórios para grande parte dos usuários, mas não todos. Usuários que necessitam de tempos de recuperação de ordem inferior a minutos para uma aplicação devem investir em modelos como o *Multi-site solution* apresentado no capítulo 4, que possuem menor tempo de recuperação, porém custo muito mais elevado por necessitarem de um ambiente replicado completo.

O tempo necessário para que uma aplicação utilizada como teste, desenvolvida em *Node.js*, esteja totalmente funcional no ambiente de *Cloud* é de 9 minutos e 10 segundos. Consideramos esse tempo aceitável, mas ainda é possível diminuí-lo com o uso de redes neurais que são capazes de realizar análises preditivas, já iniciando o *deploy* antes mesmo da aplicação se tornar indisponível. Para um acordo de 99% de *uptime* no ano, é aceitável a indisponibilidade do serviço por um tempo aproximadamente igual a 85 horas, ou 5100 minutos. Nesse período seria possível efetuar recuperação aproximadamente 567 vezes para a aplicação testada, um número superior a quantidade de dias do ano. Esse valor é variável de acordo com o tamanho e complexidade de instalação da aplicação recuperada, mas não varia muito além da ordem das dezenas de minutos.

É possível recuperar mais de uma aplicação em paralelo, sendo assim, caso ocorra um desastre que tenha impacto em diversas aplicações, todas elas tem capacidade de recuperação simultânea caso o tamanho de cada uma delas seja próximo.

O tamanho do arquivo utilizado como código fonte não reflete em muito impacto no momento do DR, apenas no momento de vinculação do arquivo a uma aplicação, não sendo este atributo um problema. Isso ocorre pois o arquivo contendo o códgo fonte também é armazenado no ambiente da Amazon, no serviço de storage chamado Amazon

S3, que é capaz de fornecer altas taxas de transmissão.

Espera-se com este trabalho que soluções mais econômicas como a de *Disaster Recovery* em ambientes de *Cloud Computing* sejam cada vez mais adotadas pelas empresas de telecomunicações que fazem uso de aplicativos web.

Referências

[1] Mysql master-slave and master-master replication. step by step configuration instructions.

[2] Amazon AWS. Aws sdk for python (boto).

[3] Amazon AWS. Aws sdk for ruby.

[4] Amazon AWS. Centro de segurança da aws, 2014.

[5] Cisco and Forbes. Collaborating in the cloud. *http://images.forbes.com/forbesinsights/StudyPDFs/cisco-cloud-report.pdf*.

[6] CompTIA. Comptia fast facts: Cloud transformation, 2014.

[7] DELL. Revealing decision points around technology adoption, use and benefits in midsize organizations, 2014.

[8] Gartner Application Architecture Development and Integration Summit. Cloud Computing as Gartner sees it.

[9] The Availability Digest. The Value of Availability. Technical report, 6 2011.

[10] Gartner. Magic quadrant for application performace monitoring, 2014.

[11] Attila Narin Glen Robinson and Chris Elleman. Using amazon web services for disaster recovery. *http://d36cz9buwru1tt.cloudfront.net/AWS$_D$isaster$_R$ecovery.pdf*.

[12] Globo.com. Entenda o caso de edward snowden, que revelou espionagem dos eua, 2014.

[13] Fábio Lucio Soares Gomes and Télio Luiz Pacheco. Missão crítica: Novos parâmetros para projetos de ti para o mercado financeiro, 2007.

[14] BT Group. Confiança das empresas na segurança de dados na nuvem é a menor já registrada, 2014.

[15] Opinion Matters. Online shoppers at slow websites.

[16] Peter Mell and Timothy Grance. The nist definition od cloud computing - recommendations of the national institute of standards and technology. *http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf*.

[17] NewRelic. Getting started with the new relic rest api (v1).

[18] Randy Perry. New Relic Yields 314% ROI for Global Technology Manugacturer. Technical report, IDC, 09 2013.

[19] K.K. Ramakrishnan Prashant Shenoy Jacobus van der Merwe Timothy Wood, Emmanuel Cecchet and Arun Venkataramani. Disaster recovery as a cloud service: Economic benefits deployment challenges. *http://lass.cs.umass.edu/papers/pdf/HC10-dr-cloud.pdf*.

[20] Manoel Veras. *Cloud Computing: Nova arquitetura da TI*. Brasport, São Paulo, São Paulo, 2012.