FACULDADE FARIAS BRITO
CIÊNCIA DA COMPUTAÇÃO

HOUNSTON DE LIMA DOS SANTOS

COMO MIGRAR APLICAÇÕES WEB CONVENCIONAL PARA AS NUVENS: UM ESTUDO DE CASO COM O GOOGLE APP ENGINE

Fortaleza
2013

HOUNSTON DE LIMA DOS SANTOS

COMO MIGRAR APLICAÇÕES WEB CONVENCIONAL PARA AS NUVENS: UM ESTUDO DE CASO COM O GOOGLE APP ENGINE

Monografia apresentada para obtenção dos créditos da disciplina Trabalho de Conclusão do Curso da Faculdade Farias Brito, como parte das exigências para graduação no Curso de Ciência da Computação.

Orientador: MSc. Murilo Eduardo Ybanez Nascimento.

Fortaleza
2013

COMO MIGRAR APLICAÇÕES WEB CONVENCIONAL PARA AS NUVENS: UM ESTUDO DE CASO COM O GOOGLE APP ENGINE

Hounston de Lima Dos Santos

NOTA: FINAL (0-10): _____

Data: ____/____/_____

BANCA EXAMINADORA

_____

MSc. Murilo Eduardo Ybanez Nascimento
(Orientador)

_____

Prof. Dr. Daniel Matos Alves
(Examinador)

_____

Prof. Me. Sérgio Araújo Yunes
(Examinador)

RESUMO

Esse trabalho analisa os problemas existentes na migração de aplicações Web em Java para o Google App Engine. Exitem muitas vantagens oferecidas pelos serviços de Nuvem, como baixo custo, manutenção mais simplificada e aquisição imediata de recurso. Em função destes fatores, começa a surgir a demanda de migrar aplicações para as Nuvens, pois assim elas podem se tornar mais competitivas. Por meio de um estudo de caso, foi realizada uma análise dos problemas existentes na migração de uma aplicação Web convencional desenvolvida em Java para o Google App Engine. Como resultados, temos um conjunto de sugestões e soluções para o desenvolvimento e migração para a plataforma do Google App Engine.

Palavras chave: Google App Engine, Migração para nuvens, Computação nas Nuvens.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

NIST	*National Institute of Standards and Technology*
ANS	*Acordos de Nível de Serviço*
SaaS	*Software as a Service*
PaaS	*Platform as a Service*
IaaS	*Infrastructure as a Service*
CRM	*Customer Relationship Management*
GAE	*Google App Engine*
Amazon EC2	*Amazon Elastic Compute Cloud*
URL	*Uniform Resource Locator*
JVM	*Java Virtual Machine*
JDO	*Java Data Objects*
JPA	*Java Persistence API*
REST	*Representational State Transfer*
API	*Application Programming Interface*
BLOB	*binary large object*
XMPP	*Extensible Messaging and Presence Protocol*
Java SE	*Java Platform Standard Edition*
Java ME	*Java Platform Micro Edition*
Java EE	*Java Platform Enterprise Edition*
CGI	*Commom Gateway Interface*
JDBC	*Java Database Connectivity*
JSP	*Java Server Pages*
HTML	*Extensible Hypertext Markup Language*

XML	*Extensible Markup Language*
JSTL	*Java Server Tag Library*
JSF	*Java Server Faces*
SQL	*Structured Query Language*
MVC	*Model View Controller*
AJAX	*Asynchronous JavaScript and XML*
JTA	*Java Transaction API*
JNDI	*Java Naming and Directory Interface*
JMS	*Java Message Service*
MOM	*Message-Oriented Middleware*
EJB	*Enterprise Java Beans*
JTA	*Java Transaction API*
UI	*User interface*
BO	*Business Objects*
MB	*ManagedBean*
MDB	*Message-driven Bean*
DAO	*Data Access Object*

1. INTRODUÇÃO

Computação nas Nuvens é um termo genérico que surgiu em 2006 em uma palestra de Eric Schmidt (executivo da Google) (TAURION, 2009) para representar a capacidade de disponibilizar recursos de TI e aplicações como serviços acessíveis remotamente pela Internet (CELLARY, Wojciech; STRYKOWSKI, Sergiusz, 2010).

Por ser um termo genérico, não é possível definir uma subárea à qual Computação nas Nuvens se enquadre. As tecnologias envolvidas no fornecimento dos serviços de Computação nas Nuvens permeiam por várias áreas da computação, como redes, bancos de dados, sistemas distribuídos, dentre outras.

Organizações como o *National Institute of Standards and Technology* (NIST), órgão norte-americano não regulatório responsável por criar padrões e medidas, tem se manifestado no intuito de tentar estabelecer padrões para Computação nas Nuvens, pois não existem métricas ou classificações consolidados para os serviços oferecidos nesta nova plataforma.

As vantagens oferecidas pela Computação nas Nuvens são mútuas, ou seja, são boas tanto para os prestadores de serviço como para quem aluga os serviços, e estão relacionadas basicamente à diminuição de custos através da racionalização no uso de recursos computacionais (ARMBRUST *et al*, 2009, p.1). Mesmo com vantagens evidentes, a Computação nas Nuvens ainda tem que resolver uma série de obstáculos que impedem sua adoção e crescimento.

Deve-se ter em mente, também, que Computação nas Nuvens não será sempre a melhor solução. Há sistemas que, por diversos fatores, nunca poderão ir para as nuvens. Tentar adequar sistemas que não foram desenvolvidos com o intuito de serem executados nas Nuvens é o grande desafio, pois os serviços de Computação nas Nuvens impõem uma série de restrições que aplicações comuns não estão acostumadas a ter.

Alguns trabalhos de VELTE *et al* (2010) norteiam até certo ponto sobre a questão da migração para as Nuvens, no entanto, ainda de forma muito superficial.

O presente trabalho está dividido em cinco capítulos. O primeiro capítulo explica de forma teórica o que é computação nas nuvens.

O segundo capítulo apresenta o Google App Engine. Nesse capítulo é explicado como funciona a arquitetura, os ambientes de execução e o armazenamento de dados do Google App Engine.

No terceiro capítulo, esclarecemos o que é uma aplicação Web desenvolvida em Java e explanamos a respeito de suas diferentes tecnologias.

No quarto capítulo, abordamos os problemas existentes na migração de aplicações para o Google App Engine e exemplificamos isso através da migração de uma aplicação web Java.

No quinto capítulo, temos as conclusões a respeito do estudo realizado.

2. COMPUTAÇÃO NAS NUVENS

Computação nas nuvens refere-se aos aplicativos negociados como serviços através da Internet e do *hardware* e *software* negociados como serviços nos *datacenters* (ARMBRUST *et al*, 2009, p.1). Dessa forma, é possível migrar o processamento local do cliente para servidores localizados nas instalações do fornecedor. Os prestadores de serviços estão preparados para cuidar da manutenção e configuração de parte dos recursos, o que torna o trabalho mais simples e possibilita redução de custos. É um retorno ao modelo dos terminais dos *mainframes*, pois o processamento real se encontra em grandes servidores e não nas máquinas locais.

Em "*A Break in the Clouds"* (VAQUERO *et al.*, 2009, p. 51), os pesquisadores dão a seguinte descrição de nuvem:

> As nuvens são um *pool* grande de fácil uso e virtualização acessível de recursos (como o *hardware*, plataformas de desenvolvimento e/ou os serviços). Estes recursos podem ser dinamicamente reconfigurados para se ajustar a uma carga variável (escala), permitindo também uma melhor utilização dos recursos. Este pool dos recursos é explorado tipicamente pelo modelo pague-por-uso em que as garantias são oferecidas pelo fornecedor da Infraestrutura por meios de ANS (Acordos de Nível de Serviço) customizados.

No modelo *pay-per-use* (pague-por-uso), o usuário paga apenas pelo que ele utilizou do serviço. Uma vez que só é pago o que é consumido, ajuda a cortar custos de quem contrata os serviços.

De acordo com o NIST, a Computação nas Nuvens é composta por cinco características essenciais, três modelos de serviço, e quatro modelos de implementação.

Existem muitos autores definindo o que é Computação nas nuvens, mas neste trabalho utilizaremos o modelo proposto pelo NIST para delimitar o que é Computação nas Nuvens.

2.1 Virtualização

Um dos componentes chave da Computação nas Nuvens é a virtualização. A

virtualização diz respeito à criação de ambientes virtuais, conhecidos como máquinas virtuais, a fim de abstrair as características físicas do *hardware*. As máquinas virtuais, por exemplo, podem ser usadas para emular diversos sistemas operacionais em uma única plataforma computacional. Assim, forma-se uma camada de abstração dos recursos dessa plataforma, alocando-se um *hardware* virtual para cada sistema (CHIRIGATI, 2009).

A virtualização permite tornar os recursos dos *datacenters* dinâmicos, de forma que os recursos entregues ao cliente possam ser reconfigurados. Isso permite a escalabilidade de recursos além de eliminar a preocupação com a camada física, uma vez que o cliente tem acesso apenas à máquina virtual e não ao *hardware* em si.

Duas técnicas de virtualização nos interessam, são elas (SUN, 2009): paravirtualização, que permite que um único servidor físico possa ser tratado como diversos servidores virtuais, e *clustering*, que permite que múltiplos servidores físicos possam ser tratados como um único servidor virtual. Na Figura 1 há uma ilustração das duas técnicas de vitualização.

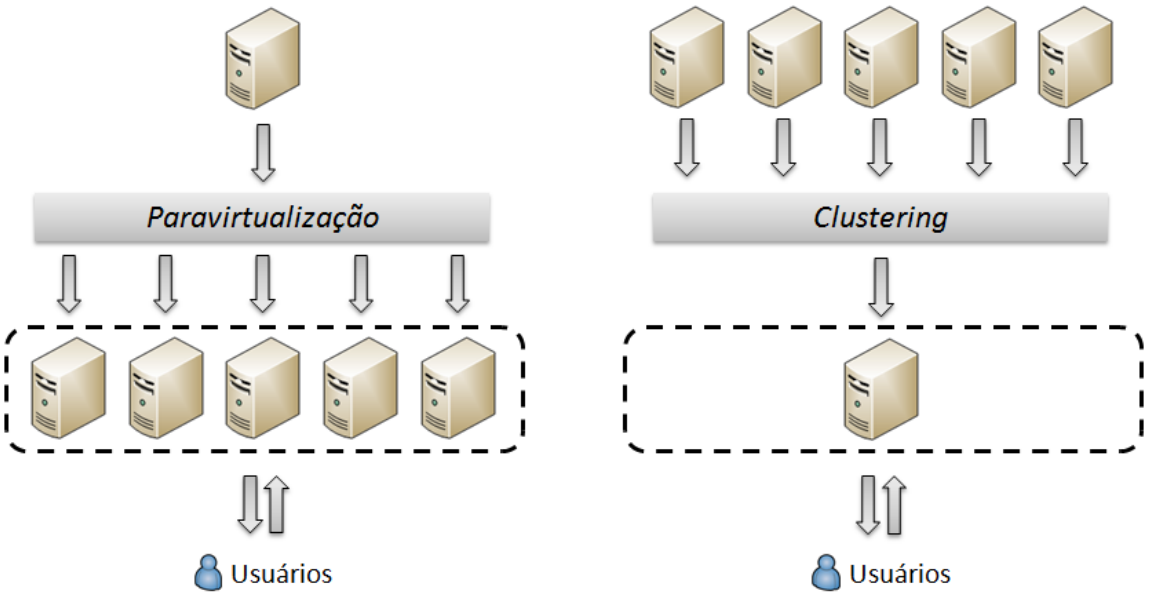

Figura 1 - Há dois tipos de virtualização que devem ser suportados na computação em nuvem: paravirtualização e clustering. (CHIRIGATI, 2009).

2.2 Características Essenciais

Conjunto de características comumente encontradas nas plataformas de Computação nas Nuvens.

a) Entrega de serviço sob demanda

O usuário pode adquirir unilateralmente recurso computacional, como tempo de processamento no servidor ou armazenamento na rede, na medida em que necessite e sem precisar de interação humana com os provedores de cada serviço (MELL, Peter; GRANCE, Timothy, 2011)[1].

b) Amplo acesso a rede

Recursos são disponibilizados por meio da rede e acessados através de mecanismos padronizados que possibilitam o uso por plataformas *thin* ou *thin client*, tais como celulares, *laptops* e PDAs. A interface de acesso à nuvem não obriga os usuários a mudarem suas condições e ambientes de trabalho, como por exemplo, linguagens de programação e sistema operacional (SOUSA, Flávio; MACHADO, Leonardo, 2010).

c) *Pooling* de recursos

O provedor de recursos é estruturado para atender vários usuários através de um modelo *multi-tenant* (capacidade de atender a múltiplos usuários usando uma única instância de aplicação no servidor), com diferentes recursos físicos e virtuais atribuídos dinamicamente de acordo com a demanda do consumidor. Há um senso de independência local em que o cliente geralmente não tem nenhum controle ou conhecimento sobre a localização exata dos recursos disponibilizados, mas pode ser capaz de especificar o local em um nível maior de abstração (por exemplo, escolher qual país, estado ou qual *datacenter* vai servir o recurso). Exemplos de recursos incluem espaço de armazenamento, processamento, memória, largura de banda de rede e máquinas virtuais (MELL, Peter; GRANCE, Timothy, 2011). Vale ressaltar que se trata aqui de um conceito distinto da virtualização.

d) Elasticidade rápida

Recursos podem ser adquiridos de forma rápida e elástica, em alguns casos automaticamente, para quando houver aumento da demanda, e para quando houver retração dessa demanda. Para os usuários, muitas vezes os recursos disponíveis para uso parecem ser ilimitados e podem ser adquiridos em qualquer quantidade e a qualquer momento (MELL, Peter; GRANCE, Timothy, 2011). O usuário deve ter a ilusão de recurso ilimitado e, no

[1] Tradução de Fabrício Silva (2009)

momento oportuno, precisa ser capaz de utilizar o recurso que ele necessita no momento. O que ajuda muito na característica de elasticidade rápida na Computação nas Nuvem é a virtualização (SILVA, Fabrício, 2009) e os serviços estruturados no modelo *multi-tenant.*

e) Serviço medido

Sistemas em nuvem automaticamente controlam e otimizam o uso de recursos por meio de uma capacidade de medição. A automação é realizada em algum nível de abstração apropriado para o tipo de serviço, tais como armazenamento, processamento, largura de banda e contas dos usuários ativas (SOUSA, Flávio; MACHADO, Leonardo, 2010). Os usos de recursos podem ser monitorados, controlados e reportados, fornecendo transparência para ambos, fornecedor e consumidor, a respeito da utilização do serviço (MELL, Peter; GRANCE, Timothy, 2011).

2.3 Modelos de Serviço

Forma pela qual os serviços são ofertados, a saber: *Software* como Serviço, Plataforma como Serviço e Infraestrutura como Serviço.

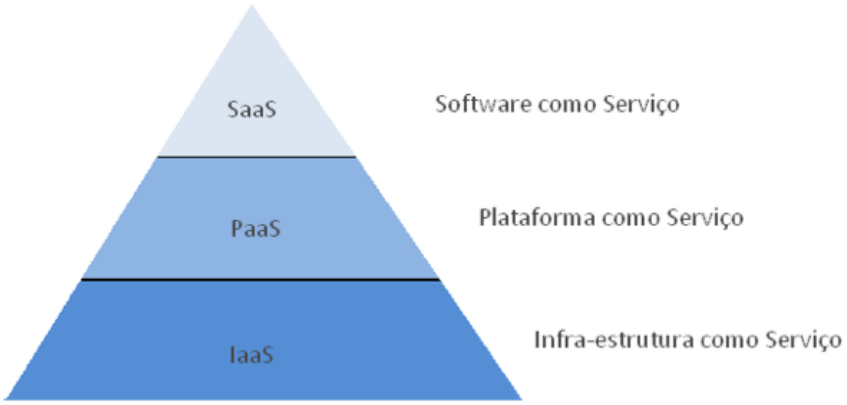

Figura 2 - Camadas de computação nas nuvens.(SILVA, Fabrício, 2009).

a) Software como Serviço (*Software as a Service* - SaaS)

Consiste na entrega de aplicações prontas para uso através da Internet. O cliente paga pelo uso da aplicação, que por sua vez é executada nos *datacenters* do provedor. Ela é a categoria mais visível aos usuários finais. Os clientes de SaaS podem ser outras aplicações.

O SaaS difere do paradigma atual no qual o usuário adquire um software e passa a ser proprietário desse software. No SaaS, o pagamento é feito por uma assinatura na qual o cliente só paga pelo que ele utilizou do serviço, como em um serviço de telefonia, por exemplo.

Fabrício Silva (2009), tem a seguinte definição de SaaS:

> Podemos dizer que o SaaS, representa os serviços de mais alto nível disponibilizados em uma nuvem. Esses serviços representam as aplicações completas que são oferecidas aos usuários. Os prestadores de serviços disponibilizam o SaaS na camada de aplicação, o que leva a rodar inteiramente na nuvem e pode ser considerado uma alternativa a rodar um programa em uma máquina local, assim o SaaS traz a redução de custos, dispensando a aquisição de licença de *softwares*. Colocamos como exemplo de SaaS, sistemas de banco de dados e processadores de textos.

O serviço é dito o de mais "alto nível", pois o SaaS se utiliza das outras camadas (PaaS e IaaS) para ser implementado. Na Figura 2, é representado o inter-relacionamento entre as camadas. Nessa representação, a camada mais acima se utiliza da camada mais abaixo para ser implementada.

O cliente de SaaS goza de uma série de vantagens, como: custo de licenciamento de *software* reduzido, acesso à versão mais atual da aplicação, auditoria de segurança e redução do custo de instalação do *software.* Além dessas vantagens, temos as vantagens advindas das características essenciais já comentadas.

Como exemplos de SaaS, temos: Google Docs, Google Talk, Google Calendar, os serviços de *Customer Relationship Management* (CRM) da Salesforce dentre outros.

b) **Plataforma como Serviço (*Platform as a Service* -PaaS)**

O PaaS tem por objetivo facilitar o desenvolvimento de aplicações destinadas aos usuários de uma nuvem, criando uma plataforma que agiliza esse processo (SILVA, Fabrício, 2009). Na figura 2, vemos que a PaaS é a camada intermediaria entre SaaS e IaaS.

A camada intermediária da pirâmide fornece plataforma de desenvolvimento ou *frameworks* como serviço. Permite configurar, reconfigurar e dispor os recursos do(s) servidor(es) conforme a demanda. Fornece todas as facilidades necessárias para suportar o ciclo de vida completo de construção e entrega de aplicações Web, sem a necessidade de *downloads* e instalações de aplicativos para desenvolvedores, gerentes de TI e usuários finais (NOGUEIRA, 2009). Essa camada fornece uma plataforma onde as aplicações são executadas, como também ferramentas para o desenvolvimento de aplicações. A aplicação não tem controle sobre a realocação sob desmanda dos recursos de *hardware*. Isso é feito pelo prestador do serviço. Se o usuário quiser, geralmente mediante a contratação, ele pode utilizar produtos do prestador que se encontram já implantados na infraestrutura do mesmo, como o serviço de usuários do Google.

Os prestadores desse tipo de serviço oferecem ao desenvolvedor um ambiente de programação e uma ou mais linguagens com um conjunto de APIs bem definidas, o que facilita a interação entre o ambiente de execução nas nuvens e as aplicações, acelera a implantação e garante a escalabilidade. (YOUSEFF, Lamia; BUTRICO, Maria; SILVA, Dilma, 2010).

Na figura 3, podemos ver a relação existente entre PaaS, SaaS e IaaS. A figura mostra que a PaaS é suportada pela camada logo abaixo, ou seja, a PaaS é implementando utilizando-se da IaaS. Conforme podemos ver na figura 3, a PaaS e a IaaS não são camadas visíveis ao usuário final como a SaaS. O usuário dessa camada é o desenvolvedor.

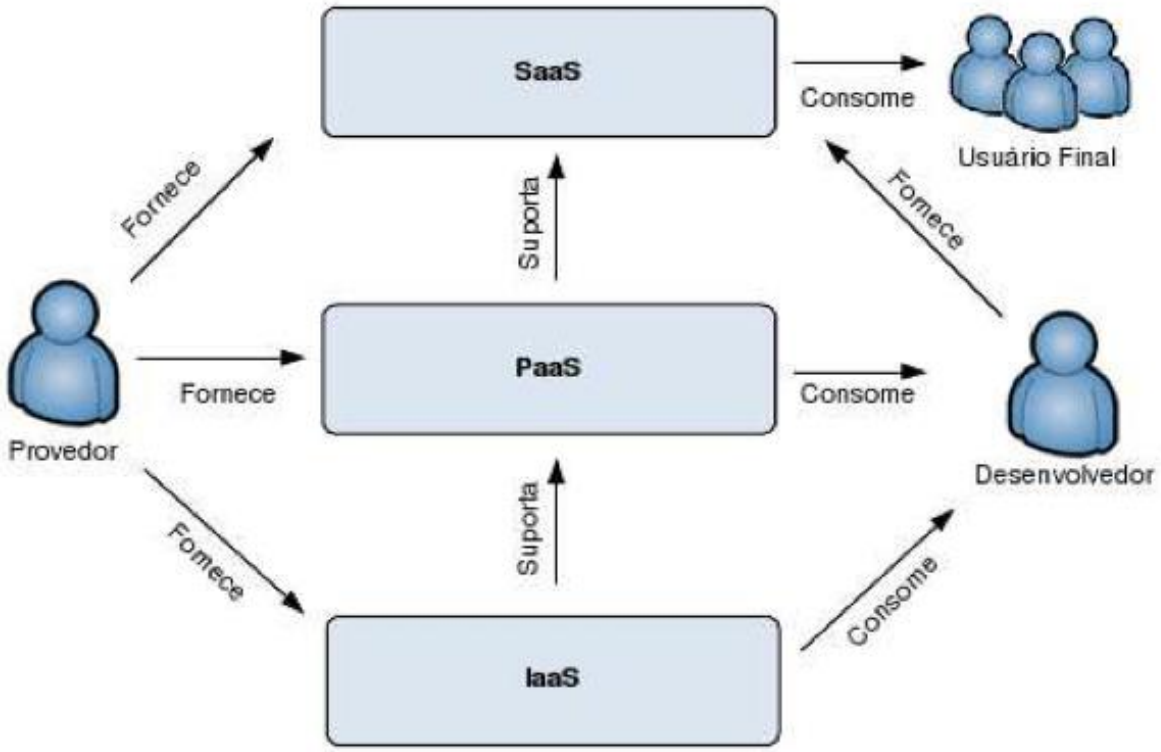

Figura 3 - Os papéis na Computação nas Nuvens (SOUSA, Flávio; MACHADO, Leonardo, 2010).

Alguns exemplos de plataformas comerciais que se encaixam nessa categoria:

- **Google App Engine:** fornece uma plataforma que permite o desenvolvimento de aplicações através da utilização das linguagens de programação Python e JAVA na infraestrutura da Google. O Google App Engine também suporta manipulação de imagens, serviços de correio eletrônico, APIs para o armazenamento dos dados, entre outros. Atualmente, a conta gratuita da ferramenta oferece um limite máximo de 1GB de armazenamento e de, aproximadamente, 5 milhões de acessos por mês (BUYYA *et al*, 2008). Caso sejam necessários mais recursos, a Google cobra pelo armazenamento e largura de banda adicional.

- **Windows Azure:** sistema operacional baseado em nuvem que permite o desenvolvimento hospedagem e gerenciamento de serviços para a Azure Services Platform. O Windows Azure disponibiliza aos desenvolvedores computação sob demanda e um ambiente de armazenamento que pode ser utilizado para hospedar, escalar e gerenciar aplicações web através de *datacenters da* Microsoft (VELTE *et al*, 2010, p.48).

c) Infraestrutura como Serviço (*Infrastructure as a Service* - IaaS)

Na Infraestrutura como serviço, conhecida pela sigla IaaS (em inglês, *Infrastructure as a Service*), a infraestrutura computacional é entregue na forma de serviço. Compreendem-se por infraestrutura recursos computacionais tais como: conexão de rede, capacidade de processamento, armazenamento de dados e *softwares* de sistema.

Através da virtualização, eles são capazes de dividir, ceder e redimensionar dinamicamente esses recursos para construir sistemas *ad-hoc* como é demandando pelo cliente (VAQUERO *et al*, 2008). Com a virtualização, eles conseguem negociar recursos de *hardware* que podem ser redimensionados quando necessário.

Os pesquisadores Wojciech Cellary e Sergiusz Strykowski (2010, p.6) provêem uma descrição bastante didática do que seria IaaS:

> Infraestrutura como um serviço consiste na entrega de infraestrutura de computadores como serviço. A infraestrutura pode incluir servidores, espaço de armazenamento, equipamentos de rede e *software* do sistema, como sistemas operacionais e sistemas de banco de dados. A infraestrutura é fornecida sob a forma de ambiente virtual. Do ponto de vista do cliente parece e funciona exatamente como uma infraestrutura padrão, enquanto na verdade ele é um dos muitos ambientes virtuais.

IaaS resolve o problema da subutilização da infraestrutura, pois o cliente contrata apenas o que necessita e, à medida que vai precisando mais, o recurso vai sendo disponibilizado sob demanda. A figura 4 mostra a utilização dos recursos de um *datacenter* local e de um *datacenter* nas nuvens. Como podemos ver, a quantidade de recursos não utilizados é bem menor e, além disso, não acontece o caso de não haver recurso para atender a demanda, algo que pode acontecer no *datacenter* local.

Dentre os serviços de IaaS existentes, o mais conhecido é o Amazon Elastic Compute Cloud (Amazon EC2). O Amazon EC2 oferece recurso computacional variável e é projetado para tornar o mais fácil possível para os desenvolvedores o redimensionamento de sua infraestrutura. Fornece uma interface Web simples que permite obter e configurar capacidades, permitindo o controle dos recursos computacionais (VELTE *et al*, 2010).

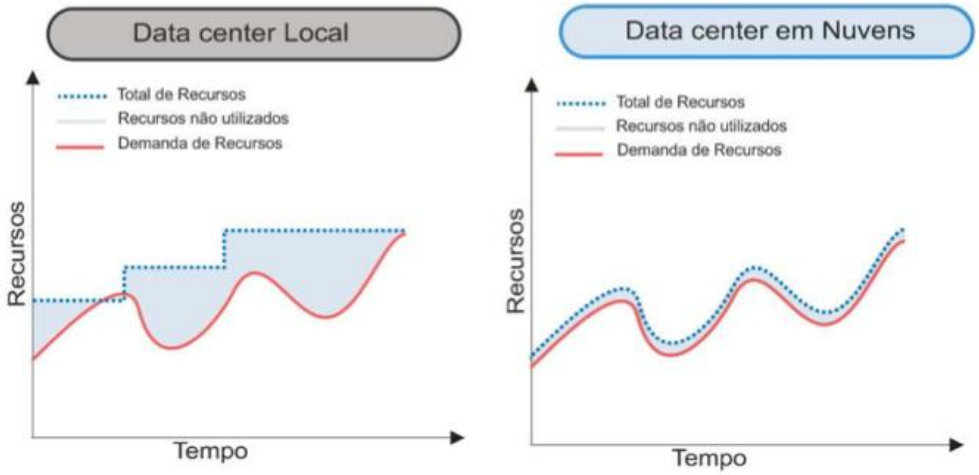

Figura 4 - Comparação de datacenter local em relação a um datacenter em *cloud computing* (SILVA, Fabrício, 2009).

2.4 Modelo de Implantação

Em se tratando do nível de acesso e disponibilidade de ambientes de nuvem, existem diferentes tipos de modelos de implantação. A restrição ou abertura de acesso depende do processo de negócio, do tipo de informação e do nível de visão. Pode-se perceber que certas empresas não desejam que todos os usuários possam acessar e utilizar determinados recursos no ambiente de sua nuvem. Neste sentido, surge a necessidade de ambientes mais restritos, onde somente alguns usuários devidamente autorizados possam utilizar os serviços providos (SOUSA, Flávio; MACHADO, Leonardo, 2010).

a) Privado

As nuvens privadas são aquelas construídas exclusivamente para um único usuário (uma empresa, por exemplo). Diferentemente de um *datacenter* privado virtual, a infraestrutura utilizada pertence ao usuário, e, portanto, ele possui total controle sobre como as aplicações são criadas na nuvem. Uma nuvem privada é, em geral, construída sobre um *datacenter* privado (CHIRIGATI, 2009).

Caso o usuário queira aumentar os recursos utilizados em sua nuvem privada, ele deve adquirir novos equipamentos, como sistemas de armazenamento, por exemplo, já que a sua nuvem está limitada à capacidade de seu sistema físico (CHIRIGATI, 2009).

Alguns autores não consideram nuvens privadas como sendo computação nas nuvens ARMBRUST (2009).

b) Comunitário

É um tipo de serviço de Computação nas Nuvens no qual várias empresas se juntam afim de montar o seu serviço. Empresas que não consigam montar sozinhas sua nuvem privada, podem se unir e montar uma nuvem de comunidade. Faz-se necessário, claro, que as empresas tenham necessidades semelhantes, de forma que a nuvem possa ser partilhada por todas as participantes (MELL, Peter; GRANCE, Timothy, 2011).

Os benefícios são o custo de criação e manutenção reduzido para os donos da nuvem, pois os gastos são divididos conforme o acordo fechado pelo grupo de empresas participantes da comunidade.

c) Público

A infraestrutura de nuvem é disponibilizada ao público em geral ou grupo de uma grande indústria e é possuída por uma organização que vende ou disponibiliza de graça serviços nas nuvens (MELL, Peter; GRANCE, Timothy, 2011).

Um dos benefícios das nuvens públicas, em geral, é que elas são maiores do que as nuvens privadas, sendo assim permitem uma maior escalabilidade dos recursos. Essa característica evita a compra de equipamentos adicionais para resolver alguma necessidade temporária, deslocando os riscos de infraestrutura para os prestadores de serviço (CHIRIGATI, 2009).

d) Híbrido

A infraestrutura é a composição de duas ou mais nuvens (privada, comunitária ou pública) que se mantém como uma única entidade através da padronização ou de tecnologia proprietária que permita a portabilidade da aplicação (MELL, Peter; GRANCE, Timothy, 2011).

As nuvens híbridas combinam os modelos das nuvens públicas e privadas. Elas permitem que uma nuvem privada possa ter seus recursos ampliados a partir de uma reserva de recursos em uma nuvem pública.

Podemos pegar como exemplo uma empresa que já possui serviço próprio de computação nas nuvens e mesmo assim se utiliza dos serviços de nuvem providos pelo Google, como hospedagem de uma aplicação.

3. GOOGLE APP ENGINE

O Google App Engine é um tipo de serviço PaaS que permite que os desenvolvedores executem seus aplicativos na mesma infraestrutura que os aplicativos próprios da Google (ZAHARIEV, 2009). Serve também para hospedar conteúdo de sites, como documentos e imagens, mas o ambiente é especialmente projetado para aplicações dinâmicas (SANDERSON, 2010, p.2).

Inicialmente, abordaremos a arquitetura do Google App Engine. Em seguida, descreveremos o Google App Engine decompondo-o em três partes: ambiente de execução, armazenamento de dados e serviços escaláveis (SANDERSON, 2010).

3.1 Arquitetura do Google App Engine

A arquitetura do Google App Engine, como podemos ver na figura 5, é constituída de um *frontend*, servidores de arquivos estáticos, servidores de aplicação, aplicação mestre, armazenador de dados, e serviços.

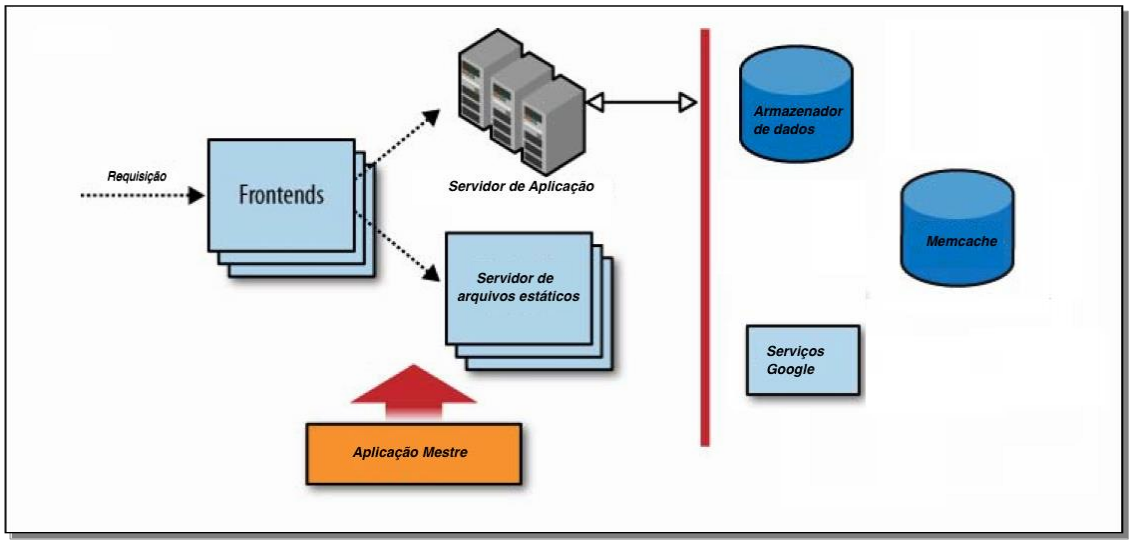

Figura 5[2] - Elementos da arquitetura do Google App Engine e seus relacionamentos.

O *frontend* promove o balanceamento de carga que distribui requisições de forma ótima, todas as requisições passam primeiras por ele.

[2] Alteração da figura encontrada na página 64 do livro *Cloud Computing : A pratical approach*

As configurações da aplicação informam ao *frontend* de que forma ele deve tratar as requisições com base nas URLs. A URL pode mapear arquivos estáticos, como imagens e textos, ou códigos da aplicação que estão aptos a tratar requisições.

No caso da URL mapear arquivos estáticos, o *frontend* encaminha a requisição para servidores de arquivos estáticos. Esses servidores são preparados para servir arquivos estáticos, possuindo topologia de rede e comportamento de *cache* para armazenar e servir de forma rápida arquivos que não mudam com frequência.

Já no caso da URL mapear um código que trate requisições, o *frontend* manda a requisição ao servidor de aplicação. O *pool* de servidores inicializa uma nova instância do aplicativo em um servidor ou reutiliza uma instância que já esteja em execução. O servidor de aplicação cuida dos recursos para a execução da aplicação, como ciclo de execução, memória e tempo de execução. O servidor de aplicação cuida também para que as aplicações não consumam recursos de forma a interferir em outras aplicações.

O código da aplicação é então executado em um dos ambientes de execução existentes. Durante a execução pelo servidor de aplicação, as aplicações podem usar os serviços disponibilizados pelo Google App Engine. Quando a execução está completa, o servidor de aplicação retorna o conteúdo gerado pelo código executado.

O *frontend* cuida de adequar a resposta ao cliente. Por exemplo, se o cliente solicita uma resposta no formato gzip, então o *frontend* cuida de compactar a resposta para esse formato.

A aplicação mestre cuida do *frontend*, servidores de arquivos estáticos e servidores de aplicação. Ele é responsável por alterações nas configurações padrões e implantação de novas versões de aplicação servidas ao usuário.

3.2 Ambiente de Execução

As aplicações do Google App Engine respondem a requisições web. Cada requisição acontece em um ambiente seguro que fornece acesso limitado ao sistema operacional. Esse ambiente é chamado de *Sandbox* (GOOGLE APP ENGINE, 2013). Tais limitações são necessárias para impedir que as aplicações que rodam no mesmo servidor

possam interferir de alguma forma entre si (SANDERSON, 2010). O *Sandbox* isola o aplicativo em seu próprio ambiente seguro e confiável, independentemente de *hardware*, sistema operacional e localização física do servidor Web (GOOGLE APP ENGINE, 2013). Este isolamento impede até mesmo que uma requisição de uma aplicação enxergue outra requisição simultânea da mesma aplicação.

Algumas das restrições impostas pelo *Sandbox* são (SANDERSON, 2010):

- As aplicações não podem criar processos ou *threads*. Todo o processamento de uma requisição deve ser realizado por um processo manipulador de requisições.

- As aplicações não podem criar conexões de rede arbitrárias. Alguns serviços de rede são fornecidos pelo Google App Engine, como URL *Fetch* e E-mail.

- As aplicações só podem ler seus arquivos e código. Não podem criar ou modificar arquivos. Em vez de arquivo, as aplicações podem usar armazenamento de dados.

- As aplicações não podem saber sobre os aplicativos ou processos que estão sendo executados no servidor.

Para ser suportado pelo Google App Engine, os aplicativos devem ser escritos nas linguagens de programação Java, Python, Go, Php ou qualquer outra linguagem que use interpretador ou compilador baseado na JVM, como JavaScript ou Ruby. Para esses aplicativos poderem ser executados, o Google App Engine conta com três ambientes de execução (em inglês, *runtime environment*) distintos, que são:

- **Python runtime environment**: Interpretador otimizado de Python com a biblioteca Python padrão.

- **Java runtime environment**: Ambiente de execução em Java do Google AppEngine.

- **Go runtime environment**: Ambiente que executa nativamente código Go compilado.

- **Php runtime environment**: Ambiente de execução experimental de Php.

Cada um desses ambientes de execução é um *Sandbox*, de tal forma que qualquer tentativa de usar um recurso da linguagem ou biblioteca que requerem acesso fora do *Sandbox* falhará, resultando em uma exceção. (SANDERSON, 2010)

3.3 Armazenamento de Dados

O Google App Engine fornece três poderosos serviços de armazenamento de dados distribuídos. São eles: App Engine Datastore, Google Cloud SQL e Google Cloud Storage (GOOGLE APP ENGINE, 2013).

Logo abaixo, segue uma descrição mais detalhada de cada um dos serviços de armazenamento.

a) App Engine Datastore

Não se tratando de um banco de dados relacional, o App Engine Datastore mais se assemelha a um banco de dados orientado a objetos (SANDERSON, 2010).

Os dados são divididos em uma ou mais entidades. Uma entidade possui um ou mais propriedades, que são definidas por um nome e um valor, sendo este último de um dos tipos primitivos suportados pelo banco de dados. As entidades também possuem um tipo. Entidades do mesmo tipo podem ter propriedades diferentes e propriedades com o mesmo nome, mas com tipos diferentes. Para melhor compreensão, pode-se imaginar o tipo da entidade como sendo uma superclasse, e as entidades em si definindo subclasses. Dessa forma, podemos considerar as propriedades como sendo “herdadas” e/ou sobrescritas, como em uma linguagem orientada a objetos.

As entidades não possuem esquema. A estrutura das entidades é fornecida e aplicada pelo código da aplicação. As interfaces Java JDO/JPA e a interface de armazenamento de dados em Python incluem recursos para aplicar e criar a estrutura de dados em sua aplicação. O aplicativo também pode acessar o armazenamento de dados diretamente para aplicar as mudanças necessárias (GOOGLE APP ENGINE, 2013).

Na Figura 6 é possível vermos a inclusão no banco de dados da entidade Employee através de código Java.

```
DatastoreService datastore = DatastoreServiceFactory.getDatastoreService();


Entity employee = new Entity("Employee");

employee.setProperty("firstName", "Antonio");
employee.setProperty("lastName", "Salieri");

Date hireDate = new Date();
employee.setProperty("hireDate", hireDate);

employee.setProperty("attendedHrTraining", true);


datastore.put(employee);
```

Figura 6 - Exemplo de criação e inserção de uma entidade utilizando Datastore. Código em Java (GOOGLE APP ENGINE, 2013)

Os serviços deste banco de dados executam transações que garantem atomicidade nas operações de manipulação de dados. Todas as operações de inserção, atualização e exclusão de uma entidade acontecem em uma transação, porém é possível com uma única transação ler e alterar múltiplas entidades. No entanto, o App Engine Datastore deve ser informado disso no momento da criação das entidades através da criação do *entity group* (grupo de entidades). Toda entidade pertence a um grupo de entidade, que é um agrupamento lógico criado para facilitar a execução de operações que de alguma forma estejam relacionadas. Por exemplo, se uma entidade Cliente será excluída, e esse cliente possui um ou

mais endereços, é ilógico não excluir seus endereços, sendo assim, informando o grupo da entidade, é possível na mesma transação excluir o cliente e os endereços.

O sítio do Google App Engine[3] descreve de forma muito clara como funciona a concorrência do Google App Engine:

> O armazenamento de dados é altamente consistente e usa controle de concorrência otimista. Uma atualização de entidade ocorre em uma transação com um número fixo de tentativas, caso outros processos estejam tentando atualizar a mesma entidade simultaneamente. Seu aplicativo pode executar diversas operações de armazenamento de dados em uma única transação, sendo que todas terão sucesso ou falharão, assegurando assim a integridade de seus dados.

Concorrência otimista é o método que assume que múltiplas transações podem ser completadas sem se afetarem entre si, dessa forma não há necessidade de bloquear o dado no qual se está operando.

b) Google Cloud SQL

Google Cloud SQL é um serviço web que permite a criação, configuração e o uso de bancos de dados relacionais nas nuvens do Google. É um serviço completamente gerenciado e administrado pelo provedor do serviço, o que permite que o usuário se foque unicamente nas questões concernentes a suas aplicações e seus serviços. (GOOGLE APP ENGINE, 2013).

É um banco muito similar ao MySQL, o que torna fácil a migração de um banco de dados relacional preexistente (GOOGLE APP ENGINE, 2013).

Suas únicas limitações são tamanho das instâncias (100GB), criação de funções (não suportado), além de alguns comandos, como (GOOGLE APP ENGINE, 2013):

- `LOAD DATA INFILE`
- `SELECT ... INTO OUTFILE/DUMPFILE`
- `INSTALL/UNINSTALL PLUGIN`

[3] https://developers.google.com/appengine/ acesso feito em 28/11/2011

- `CREATE FUNCTION`
- `LOAD_FILE()`
- `SHA2()`

c) Google Cloud Storage

Serviço RESTfull de armazenamento de dados. Permite o armazenamento de blob (*binary large object*) e de arquivos de grandes tamanhos (terabytes). De acordo com a documentação do Google Cloud Storage[4] não há limites de armazenamento ou de volume de consulta.

Todos os dados no Google Cloud Storage pertencem a um projeto. O projeto possui um conjunto de usuários, um conjunto de APIs, configurações de cobrança de taxas, autenticação e monitoramento de APIs. Através da figura 7 é possível ter uma melhor compreensão a respeito desses elementos do projeto.

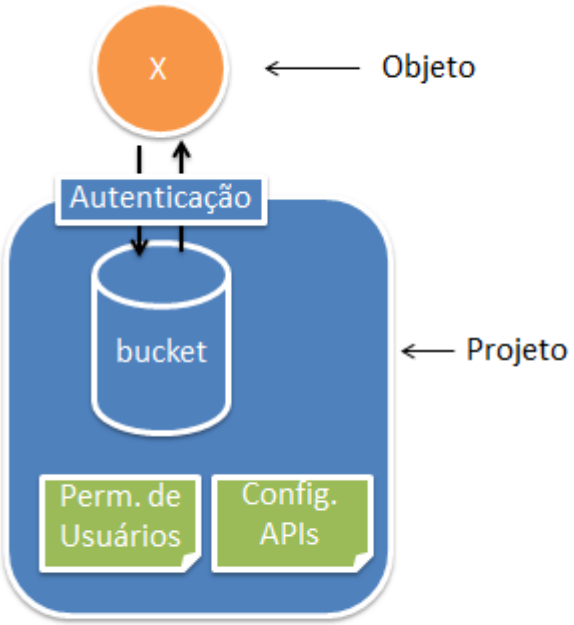

Figura 7 – Elementos do Google Cloud Storage

[4] https://developers.google.com/storage/docs/overview acessado em 11/06/2013

No Google Cloud Storage, os dados são salvos em estruturas chamadas de *Buckets*. O *Bucket* pode ser usado para organizar os dados e para controlar o acesso. O *Bucket* não pode ser compartilhado entre projetos

Os pedaços de dados salvos no Google Cloud Storage são chamados de Objetos. Esses Objetos possuem dois componentes: um dado e um *metadado.* O dado é o aquivo que quer ser salvo. O *metadado* é uma lista de valores com informações sobre o arquivo. Esse Objeto é salvo no *Bucket* e não pode ser compartilhado (GOOGLE CLOUD STORAGE, 2013).

3.4 Outros Serviços

O Google App Engine fornece vários serviços escaláveis úteis para as aplicações (SANDERSON, 2010). Alguns exemplos (GOOGLE APP ENGINE, 2013):

- **Memcache:** memória de alto desempenho. Pode ser acessada por diversas instâncias da aplicação. Foi feito para ser usado com dados que não vão ser persistidos ou para copiar dados de consultas a fim de tornar a busca mais veloz.
- **Busca de URL:** através da URL é possível ter acesso a outros serviços na Web ou a outros dados.
- **E-mail:** a aplicação pode enviar e-mails usando a infraestrutura do Google.
- **Manipulação de imagens:** com essa API é possível redimensionar, cortar, girar e inverter imagens nos formatos JPEG e PNG.
- **Contas do Google:** permite integração com o sistema de contas de usuários do Google.
- **Envio de Mensagens XMPP:** permite o envio de mensagens no formato XMPP assim como as do GTalk.
- **Task Queue:** permite a execução de tarefas em segundo plano, conforme configuração de agendamento feita pelo em código.
- **Cron:** serviço que permite o agendamento de tarefas através de configuração**.**

4. APLICAÇÕES WEB EM JAVA

O Java possui 4 plataformas de desenvolvimento: Java SE (*Java Platform, Standard Edition*), Java ME (*Java Platform, Micro Edition*), Java FX e Java EE (*Java Platform, Enterprise Edition*).

Todas as plataformas são suportadas por uma máquina virtual e um conjunto de APIs (Interfaces de Programação de Aplicações). A máquina virtual é um programa para uma plataforma de *hardware* e *software* específica, sendo responsável por executar uma aplicação Java. As APIs são uma coleção de componentes de *software* que são utilizados na elaboração de um novo *software* ou componente de *software* (JENDROCK *et al*., 2013).

No sítio do Netbeans[5], temos uma ótima descrição do que é Java EE:

> O Java EE (*Enterprise Edition*) é uma plataforma amplamente usada que contém um conjunto de tecnologias coordenadas que reduz significativamente o custo e a complexidade do desenvolvimento, implantação e gerenciamento de aplicações de várias camadas centrados no servidor. O Java EE é construído sobre a plataforma Java SE e oferece um conjunto de API's (interfaces de programação de aplicações) para desenvolvimento e execução de aplicações portáteis, robustas, escaláveis, confiáveis e seguras no lado do servidor.

Existem 3 camadas lógicas presentes em todas aplicações Java EE, são elas as camadas de: apresentação, lógica de negócio e acesso a base de dados. Na figura 8, temos uma típica arquitura Java EE de três camada.

[5] Um dos principais ambientes de desenvolvimento Java, mantido pela própria proprietária da linguagem. https://netbeans.org/kb/trails/java-ee_pt_BR.html acesso feito em 01/06/2013

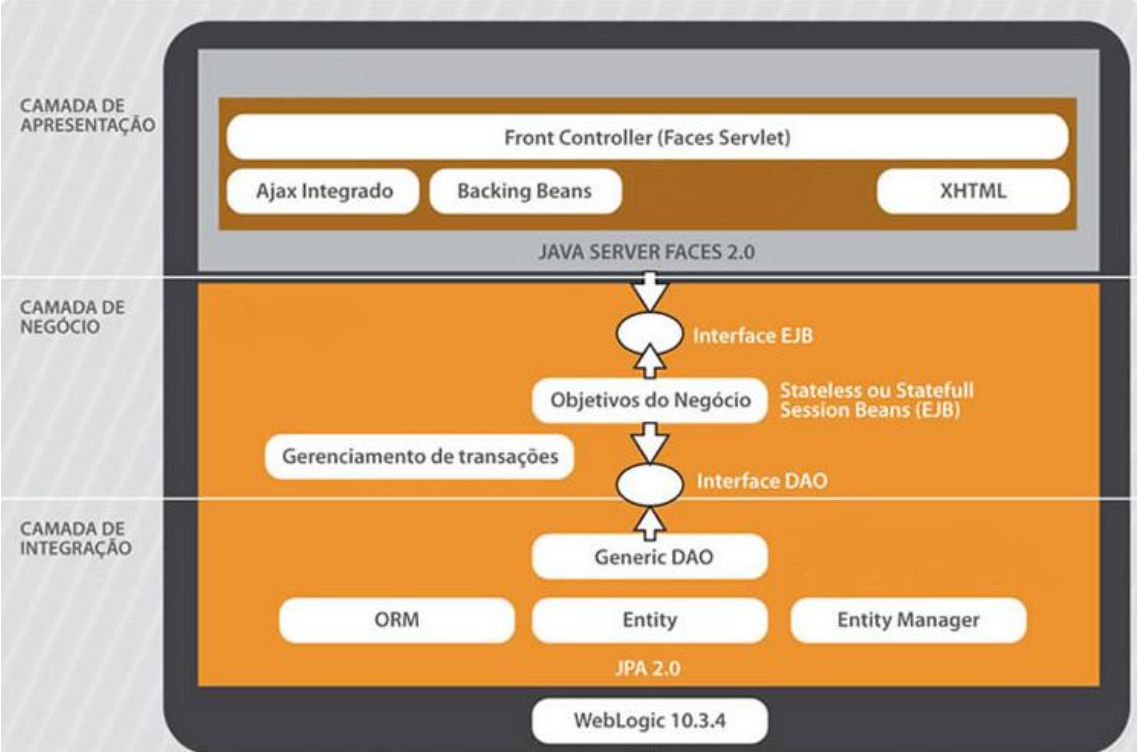

Figura 8[5] – Arquitetura típica de uma aplicação Java EE. Essa aplicação utiliza JSF, EJB e JPA.

A camada de apresentação é responsável por exibir os dados ao usuário e chamar o componente de negócio apropriado quando são gerados eventos de interface. A lógica concernente a essa camada é apenas de exibição e controle de dados. A camada de negócio é responsável pela lógica da aplicação, ou seja, a lógica relacionada ao negócio do sistema. A terceira camada é a responsável por abstrair para a camada de negócio o acesso à base de dados. Ela torna a conexão com a base de dados transparente para aplicação, possibilitando o desacoplamento entre a aplicação e o Banco.(MUKHAR, Kevin; ZELENAK, Chris, 2005) Essa divisão em camadas permite o desacoplamento da aplicação e suas partes, além de demarcar claramente o uso de cada um das APIs utilizadas.

Logo abaixo, são apresentadas algumas das APIs mais importantes do Java EE:

- **API Java Servlet:** tecnologia da plataforma Java que viabiliza aplicações web sem as limitações de desempenho do CGI (*Commom Gateway Interface*). Ao

[5] Alteração da imagem encontrada no sítio da empresa gaúcha de desenvolvimento Voiza. Original cncontra-se em http://www.voiza.com.br/?secao=fabrica-software-tecnologias.

contrário dos mecanismos de extensão de servidores proprietários (como os módulos do Apache), Servlets são independentes de plataforma. Isso significa que se está livre para escolher a melhor estratégia em servidores, plataformas e ferramentas. Além disso, *servlets* tem acesso a toda a família de APIs Java, incluindo a API JDBC para acessar bancos de dados corporativos (JENDROCK *et al*., 2013).

- **Java Server Pages (JSP)**: JSP é uma extensão da API Java Servlet, criada para apoiar a criação de páginas HTML e XML. Torna mais fácil o desenvolvimento, pois combina dados fixos ou estático com o conteúdo dinâmico da página. De forma resumida, são páginas de texto que são processadas em um *servlet* e produzem um conteúdo dinâmico (JENDROCK *et al*., 2013).

- **Java Server Tag Library (JSTL)**: conjunto de *tags* JSP que encapsulam funcionalidades que são comuns em várias aplicações web. JSTL tem suporte para tarefas comuns e estruturais, tais como iterações e condicionais, além de *tags* para manipular documentos XML, *tags* de internacionalização e *tags* SQL. Ele também permite a integração de *tags* personalizadas com as *tags* existentes JSTL (JENDROCK *et al*., 2013).

- **Java Server Faces (JSF):** JSF é uma tecnologia de desenvolvimento de páginas para a web que possui características de um framework MVC (*Model View Controller*) e de um modelo de interfaces gráficas baseado em eventos. Através do uso do padrão MVC, promove uma clara separação entre visualização e regra de negócio.(PITANGA, Talita, p.2). O JSF possui um *servlet* de controle chamado `FacesServlet`, esse *servlet* é responsável por receber a requisição da WEB, redirecioná-la a classe apropriada e remeter a resposta. No seu funcionamento, conta com arquivos de configuração que são capazes de fazer associações e mapeamentos de ações através das regras de navegação. Os manipuladores de eventos cuidam do recebimento dos dados vindos da camada de

visualização e são capazes de executar classes da camada de negócio. O resultado do que é executado na camada de negócio é devolvido pelo `FacesServlet` como resultado. (Pitanga, Talita, p.3). O JSF também possui componentes como o Richfaces que permite o uso de AJAX (*Asynchronous JavaScript and XML)* e de uma interface visual mais rica. (SHIRAH, Joe , 2008)

- **Facelets:** Facelets é uma poderosa e leve linguagem usada na criação de páginas JSF baseadas em um estilo de *template* HTML. Através do seus *templates* ele possibilita um maior reúso de código (JENDROCK *et al*., 2013).

- **Java Transaction API (JTA)**: API responsável pelo controle de transações no Java. Através de uma interface padrão essa API possibilita a abstração da real implementação da transação. Diferentes servidores de aplicação implementam transações de diferentes formas, por isso a necessidade de uma API como essa, pois possibilita tornar o código mais portável e independente (JENDROCK *et al*., 2013).

- **Java Naming and Directory Interface (JNDI)**: JNDI é uma API Java que provê nomes e funções de diretório. O serviço JNDI permite a associação de um nome a um determinado recurso, como (endereços de memória, de rede, de serviços, objetos e referências). As principais funções da JNDI e a de associar um valor a um nome e localizar um valor através de um nome. (PIRES, Paulo, 2003)

- **Java Message Service (JMS):** JMS é uma API que define um grupo de interfaces e semântica que permite que aplicações Java acessem serviços do tipo *Message-Oriented Middleware* (MOM). O MOM é um *middleware* (mediador) que permite a integração das operações de troca de mensagens intra e entre aplicações (PEREIRA, Austeclynio, 2005)

- **Enterprise Java Beans (EJB):** escrito na linguagem de programação Java, um *enterprise bean* é um componente do lado do servidor que encapsula a lógica de negócios de uma aplicação. A lógica do negócio é o código que cumpre o propósito da aplicação. Em um aplicativo de controle de estoque, por exemplo, os *beans* corporativos podem implementar a lógica de negócio em métodos(JENDROCK *et al*., 2013). Ao invocar esses métodos, os clientes podem acessar os serviços de inventário fornecidos pelo aplicativo. A API EJB permite o desenvolvimento rápido e simplificado de aplicações distribuídas, transacionais, seguras e portáteis baseadas na tecnologia Java. De forma resumida, existem dois tipos de EJB (JENDROCK *et al*., 2013):

    I. **Session Beans:** o Session Bean encapsula a lógica de negócio que pode ser chamada através de código por um cliente, seja esse cliente local, remoto ou serviço Web. Para acessar um aplicativo que é implantado no servidor, o cliente invoca os métodos do Session Bean e ele realiza o trabalho para o seu cliente, diminuindo a complexidade da execução de tarefas de negócios dentro do servidor. Além disso, os Session Beans podem ser divididos em três tipos diferentes: *stateful*, *stateless*, and *singleton*.

    II. **Message-Driven Beans:** um Message-driven bean é um EJB que permite que aplicações Java EE processem mensagens de forma assíncrona. Este tipo de *bean* normalmente atua como uma mensagem JMS ouvinte, semelhante a um evento ouvinte, mas em vez de eventos, recebe mensagens JMS. As mensagens podem ser enviadas por qualquer componente Java EE (um aplicativo cliente, outro EJB, serviço Web) ou por um aplicativo JMS ou sistema que não usa a tecnologia Java EE. Message-Driven Beans processam também outros tipos de mensagens.

- **API Java para Web Services RESTful:** a API Para Web Services RESTful (JAX-RS) define uma API de desenvolvimento de serviços Web construídos de

acordo com o estilo de arquitetura REST (*Representational State Transfer*, Transferência de Estado Representativo). REST é um estilo de arquitetura direcionado para sistemas de hipermídia distribuídos. É um estilo constituído basicamente por dois papéis: Cliente e Servidor. (LIMA, Jean, 2012)

- **Java Persistence API (JPA):** a JPA é a solução padrão de persistência do Java EE. Usa um mapeamento objeto relacional para preencher a lacuna existente entre os objetos Java e o banco de dado relacional (JENDROCK *et al*., 2013). O objeto é mapeado através de arquivos XML de configuração ou anotações em seus atributos, propriedades e classe. Essas anotações possibilitam um espelhamento do banco nos objetos e vice-versa (JENDROCK *et al*., 2013). O JPA pode ser usado por aplicações Java SE.

As aplicações Web em Java são desenvolvias usando um subconjunto de APIs (API Java Servlet, JSP, JSF e JSTL) pertences a plataforma Java EE. Comumente essas aplicações Web usam também APIs como JPA e JTA.

5. MIGRAÇÃO DE UMA APLICAÇÃO WEB PARA GOOGLE APP ENGINE

Muitos são os problemas existentes quando se fala em migração de aplicações Java para o Google App Engine(GAE), pois o mesmo impõe várias restrições aos seus aplicativos. O primeiro problema advém do fato do Google App Engine não suportar muitas das APIs da plataforma Java EE, o que dificulta as migrações das soluções que já foram feitas utilizando-se dessas APIs. Em "Comparação do PaaS em Java" (YUAN, Michael, 2011) temos um exemplo de problema com a migração do JSF e as possíveis implicações de soluções que usam de alterações do código da API para solucionar o erro:

> Essas limitações de API's impõem desafios quando há necessidade de usar *frameworks* de aplicativos já existentes ou mover aplicativos já existentes para o GAE. Depois de anos de evolução, o desenvolvimento corporativo em Java é muito dependente de *frameworks*. Embora alguns *frameworks* bastante conhecidos, como o Spring e o Struts, funcionem no GAE de fábrica, muitos outros não funcionam ou requerem *patches* no código de origem. *Hackear* manualmente o código de origem do framework para fazer com que ele execute no GAE nunca é uma boa idéia, porque você está basicamente criando uma bifurcação que quebra a compatibilidade de envio e pode apresentar erros difíceis de depurar no framework. Um bom exemplo é o *framework* do Web Java Server Faces (JSF): ele requer o *hackeamento* no nível do código de origem para executar no ambiente do GAE e, mesmo assim, muitas bibliotecas de UI baseadas no JSF são incompatíveis com o GAE

O segundo problema vem do fato do Google App Engine ter sérias limitações nas suas próprias APIs. Por exemplo, a API de acesso à rede do Google App Engine impõe limite de tempo de conexão, forçando a mesma a ser fechada depois de 5 ou 10 segundos (YUAN, Michael, 2011). O terceiro problema diz respeito ao desempenho do Google App Engine. O mesmo fornece escalabilidade excelente, medida por um tempo de resposta consistente, entretanto, o seu desempenho bruto é frequentemente lento (YUAN, Michael, 2011). Isso significa que o clock de processamento oferecido não é elevado, mas connstante e escalável. Outro problema relacionado ao desempenho é o fato do Google App Engine trocar sua aplicação de JVM quando ele notar que essa JVM está ociosa. Isso faz com que sua aplicação tenha que ser carregada quando for feita uma requisição logo após ter sido mudada de JVM, o que leva um tempo maior que o normal.

Aplicação Web em Java – Plataforma JEE

No estudo de caso, foi utilizada uma aplicação Java Web dividida em três camadas, aos moldes de uma aplicação Java EE. Essa aplicação foi feita com o intuito de simular uma aplicação Web convencional feita em Java, por isso foram utilizadas as APIs Java mais comuns. Nessa aplicação, estão presentes recursos tecnológicos recorrentes a muitas aplicações Java, como AJAX, serviço assíncrono de mensagem, processamento *batch* (Acões executadas sem intervenção direta do usuário) (SILVA; MACHADO, 2013) e acesso a banco de dados relacional via JPA. A descrição funcional da aplicação se encontra no apêndice A. Na figura 9, podemos ver o relacionamento entre cada um dos componentes da aplicação. É possível ver que os Managed Beans (MB) executam métodos nos objetos de negócio (*Bussiness Objects* - BO) que por sua vez, executam métodos dos DAOs (*Data Access Object*, ou Objeto de Acesso a Dados). Vemos também como os Message-driven Beans (MDB) enviam mensagens aos serviços REST e aos ManagedBeans.

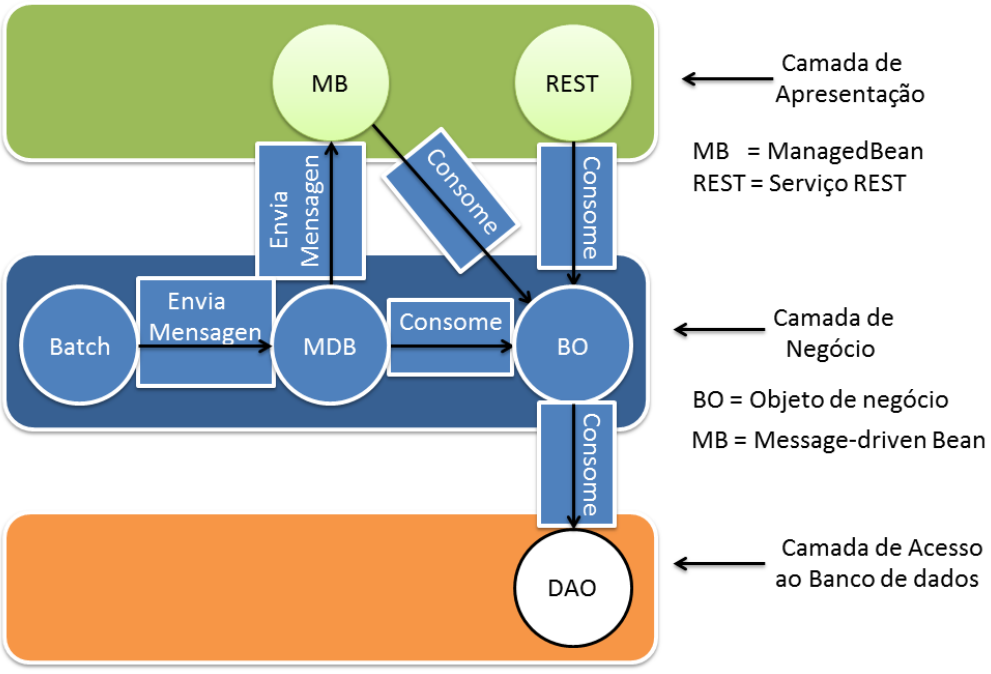

Figura 9 - Arquitetura da aplicação web em Java.

Logo abaixo, segue a descrição detalhada das características técnicas de cada camada da aplicação:

a) Apresentação

A camada de apresentação foi construída com JSF, Facelets e Richfaces. O Facelets foi usado para definir um *template* de página na qual as páginas escritas em JSF são renderizadas. Richfaces foi utilizado na exibição do status do processo *batch* que é realizado na camada de negócio. A camada de negócio executa o processo batch e envia uma mensagem assíncrona à camada de apresentação que a exibe de forma também assíncrona

b) Negócio

A camada de negócio foi construída com EJB e JMS. Nessa camada todos os objetos de negócio são EJBs que usam nativamente as APIs de JTA e JNDI. Além disso, através da funcionalidade de agendamento presente na API da EJB, foi implementado um processo *batch*.

c) Acesso a base de dados

O Acesso a base de dados se dá mediante aos DAO que internamente usam a API de persistência Java. DAO é um padrão de projeto que permite separar a lógica de negócio da lógica de persistência de dados (SULIVAN, Sean, 2003). Esses DAOs além de receber um objeto que deve ser persistido recebe também um *Entitymanager*. O *Entitymanager* (Gerenciador de entidades) é um serviço da API JPA que cuida de toda a persistência de dados.

5.1 Aplicação Web em Java – Plataforma Google App Engine

Durante a migração da aplicação para o Google App Engine, foram evitados ao máximo modificações no código da aplicação de origem. Entretanto, em alguns momentos, houve necessidade de algumas intervenções no código da aplicação, como a que é possível de ser visto na figura 10. Em alguns outros casos, foi necessário a criação de classes modificadas com o mesmo nome de classes nativas de APIs, permitindo suprimir métodos que causassem problemas.

```java
@Override
public void inserir(Usuario usuario) throws NegocioException {
    dao = new UsuarioDao(em);
    dao.create(usuario);
}
```

```java
@Override
public void inserir(Usuario usuario) throws NegocioException {
    dao = new UsuarioDao(em);
    em.getTransaction().begin();
    dao.create(usuario);
    em.getTransaction().commit();
}
```

Figura 10 – Exemplo de método antes e depois da migração.

Em outro caso, para que fosse possível a utilização de funcionalidades como agendamento da execução de um código, foi necessário a criação de um serviço REST que servisse de intermediário, pois serviços como o Cron não executam métodos diretamente como era feito com a função timer do EJB. Para receber mensagens XMPP também foi necessário criar um intermediário, nesse caso um *servlet.*

Esses tipos de alteraçãos (criação de um serviço e de um servlet intermediário), fez com que funcionalidades do sistema saíssem da camada de negócio e fossem para a camada de apresentação. Na figura 11 é possível ver a arquitetura final do sistema migrado. Os círculos em laranja(XMPP e Cron) substituíram respectivamente o timer EJB e o Message-driven Bean.

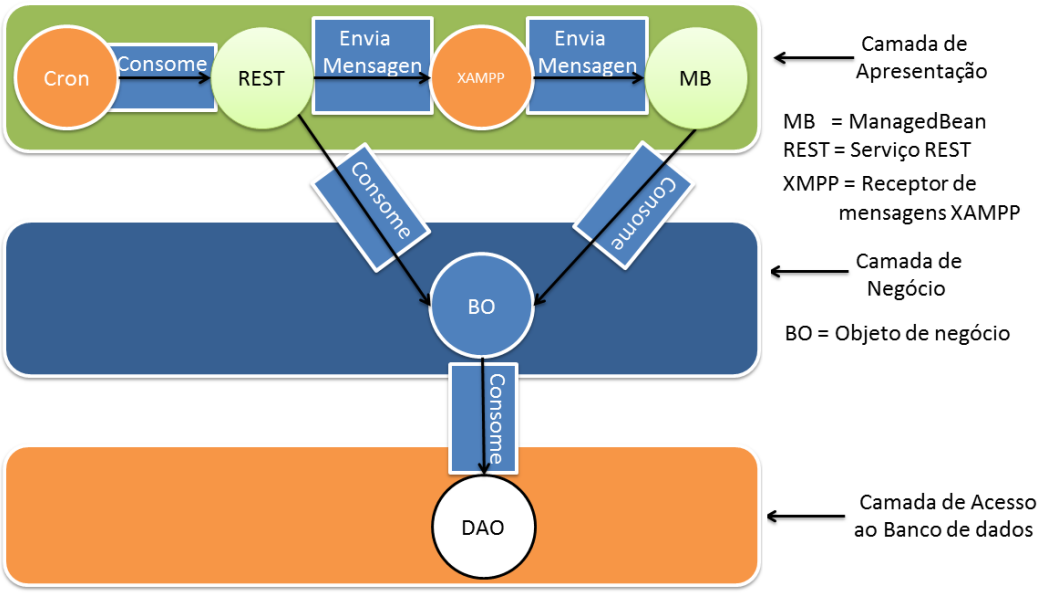

Figura 11 – Arquitetura da aplicação após a migração.

Depois da migração as camadas ficaram da seguinte forma:

a) Apresentação

Na camada de apresentação, foi necessária a substituição de duas classes do JSF que quebravam as restrições do Google App Engine

- **`com.sun.faces.application.resource.ResourceImpl:`** para corrigir uma exceção que acontecia no momento da criação do EntityManager da camada de acesso à base de dados. A classe modificada foi encontrada na página de discussão de erros do Google App Engine[6].

- **`com.sun.faces.config.WebConfiguration:`** para poder executar o JSF no Google App Engine, os métodos `processJndiEntries` e `canProcessJndiEntries` tiveram que ser comentados na classe original. No sítio do Google App Engine não há menção há esse erro. Essa solução foi encontrado no sítio de tutoriais Mkyong[7].

Também foi necessário a instanciação manual das classes de negócio, como pode ser visto na figura 12. Com EJB isso era feito automaticamente, sendo necessário apenas uma anotação (JENDROCK *et al*., 2013). A instanciação manual das classes, quando não feita corretamente, pode gerar problemas como um maior acoplamento da aplicação (SILVA, Douglas, 2012). Em um cenário utilizando EJB, é o servidor de aplicação que cuida de criar o objeto apropriado. De forma, que implementação dessa interface possa muda sem impactar no código (JENDROCK *et al*., 2013).

[6] https://code.google.com/p/googleappengine/issues/detail?id=8415
[7] http://www.mkyong.com/google-app-engine/gae-jsf-javax-naming-initialcontext-is-a-restricted-class/

```
@EJB
FuncionalidadeFacadeLocal facadeFuncionalidade;
public List<Funcionalidade> getFuncionalidade() {
    try {
        return facadeFuncionalidade.getFuncionalidades();
    } catch (NegocioException e) {
        e.printStackTrace();
        return null;
    }
}
```

```
private UsuarioBO usuarioBO = new UsuarioBOImpl();
public List<Usuario> getUsuarios() {
    try {
        usuarios = usuarioBO.getUsuarios();
        return usuarios;
    } catch (NegocioException e) {
        e.printStackTrace();
        return null;
    }
}
```

Figura 12 – Código de Managedbean antes e depois da migração.

Outro problema é que para salvar, atualizar e remover dados do banco é necessário abrir a transação manualmente, o que não era necessário com o EJB, pois ele cuida desses detalhes de abertura e fechamento de transação (JENDROCK *et al*., 2013). Na figura 10 é possível ver essa abertura manual de transação no segundo método `inserir`.

b) Negócio

Nessa camada foi necessário a remoção das anotações dos EJBs, o que implicou na criação de um serviço REST para o processo *batch* (que antes era chamado pelo *timer* do EJB) e inclusão desse serviço na rotina de execução do Cron. Na figura 13a é possível ver um serviço REST e na figura 13b uma configuração de agendamento do Cron.

a)
```java
@Path("usuario/")
public class ServicoUsuario {

    @GET
    @Path("/batch")
    @Produces("application/json")
    public String batch() throws NegocioException, CacheException {…}

}
```

b)
```xml
<?xml version="1.0" encoding="UTF-8"?>
<cronentries>
  <cron>
    <url>/servico/usuario/batch</url>
    <description>Executa batch a cada 1 minuto</description>
    <schedule>every 1 minutes</schedule>
  </cron>
</cronentries>
```

Figura 13 – O serviço de agendamento do Cron.

Houve também a substituição dos serviços JMS para XMPP, o que ocasionou refatoração do código original, exigindo a criação de um *servlet* para recebimento da mensagem, o que não era necessário com JMS. Já para o armazenamento das mensagens, utilizamos Memcache.

```java
@SuppressWarnings("serial")
public class XMPPReceiverServlet extends HttpServlet {
    public void doPost(HttpServletRequest req, HttpServletResponse res)
            throws IOException {
        XMPPService xmpp = XMPPServiceFactory.getXMPPService();
        Message message = xmpp.parseMessage(req);

        String body = message.getBody();
        try {
            Cache cache = CacheManager.getInstance().
                    getCacheFactory().createCache(Collections.emptyMap());
            cache.put("status", body);
        } catch (CacheException e) {
            e.printStackTrace();
        }

    }
}
```

Figura 14 – *Servlet* responsável por receber mensagens XMPP.

c) Acesso a base de dados

Na camada de acesso a dados, foram testados dois cenários, utilizando Google Cloud SQL e Google Datastore:

- Google Cloud SQL: com esse banco, não houve alterações nos DAO ou nas entidades. A alteração de banco se tornou transparente à aplicação. Outro ponto é que foi possível realizar migração dos dados e da estrutura de um banco SQL preexistente. A migração do esquema foi feita através do *plugin* de JPA do Eclipse. Esse plugin permitiu criar as tabelas através das entidades. Também foi possível criar essas tabelas através da execução de comandos no terminal *online* do Google Cloud SQL.

- Google Datastore: Para utilizar o datastore, foi necessário a alteração das entidades, pois o banco BigTable não aceitava relacionamentos do tipo muitos pra muitos. Também foi necessário a alteração de duas bibliotecas que eram utilizadas pela aplicação, pois havia conflitos de versão: jersey-server-1.17 e asm-3.3.1. Essas bibliotecas causavam conflito com outra versão do asm, a versão 4 que era utilizada pela API usada na conexão com o datastore. Para corrigir esse conflito, foi utilizado um aplicativo java chamada jarjar[8], que usa uma expressão regular passada através de um arquivo texto para buscar ocorrências em uma biblioteca Java. Com base na expressão, o jajar é capaz de apontar a referência para a biblioteca correta. O jajar foi usado nas bibliotecas que causavam conflito, no caso, jersey-server-1.17 e asm-3.3.1. O Google Datastore garante atomicidade e concorrência na execução de suas operações. Nesse banco, as entidades e os dados só puderam ser migrados através do desenvolvimento de uma aplicação de carga, não foi encontrado qualquer ferramenta que auxiliasse nessa tarefa

5.2 Análise Qualitativa

Conforme verificado através do estudo de caso, o Google App Engine é uma plataforma bastante completa e possui um conjunto de funcionalidades que suprem as necessidades mais comuns de uma aplicação Web, entretanto, a migração de algumas APIs externas ao Google App Engine ainda apresentam soluções questionáveis, como o *hackeamento* de APISs encontrado no JSF ou as alterações nas bibliotecas jersey-server-1.17 e do asm-3.3.1.

[8] https://code.google.com/p/jarjar/

Durante a execução dos testes, o uso isolado de APIs apresentou um número de problemas menor que o uso em conjunto, o que evidencia que quanto maior o desacoplamento da aplicação, menor a chance de ocorrerem problemas como os encontrados no JSF e no Jersey. O ideal para a aplicação que pretende ir pro Google App Engine é a separação de sua camada de negócios da sua camada de apresentação através da criação de serviços que fiquem responsáveis por executar as classes de negócios. Isso reduz o acoplamento como um todo e, consequentemente, reduz a ocorrência de conflito entre as APIs. Além disso, o uso de funcionalidades como o Cron e a fila de tarefas pressupõem o uso de serviços. Na figura 15, temos como ficaria uma aplicação onde todas as requisições passavam por um serviço.

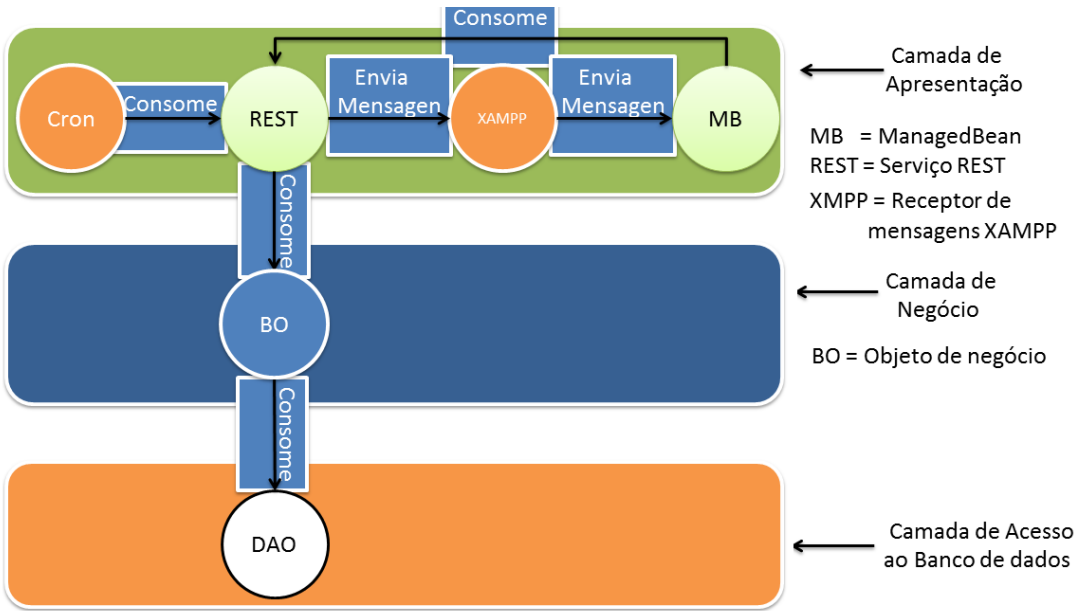

Figura 15 – Arquitetura ideal proposta.

Tendo criado esses serviços, e consequentemente desacoplado os códigos, os únicos problemas na manutenção seriam os referentes a APIs que tiveram classes *hackeadas*, pois quando essa solução é aplicada criamos uma API levemente diferente da API padrão de mesma versão, então quando houver a atualização dessa API para uma nova versão essa solução utilizada pode se tornar inválida. O Google App Engine também permite ao administrador ter acesso a diversos dados que possibilitam encontrar erros de execução e locais onde está havendo mais uso da CPU e rede.

Para auxiliar no desenvolvimento, o Google App Engine conta com um *plugin*(GOOGLE APP ENGINE, 2013) para o ambiente de desenvolvimento Eclipse que facilita

diversas ações, como: implantação no Google App Engine e configuração de conexão com banco(Google Cloud SQL e Datastore).

Quanto ao desempenho, através do estudo de caso concluimos que o Google App Engine não possui um desempenho bruto elevado, ele é escalável e possui um desempenho razoável, demorando em média o mesmo tempo de resposta que a aplicação demorava quando era executada em um servidor de aplicação instalado localmente.

6. CONCLUSÃO

Apesar de trazer muitas vantagens, ainda é complexo mudar do modelo de computação convencional para a Computação nas Nuvens. Muitas soluções de computação convencional já foram feitas, e criar novas soluções que se adéqüem a Computação nas Nuvens pode ser muito caro e demorado, ao passo que é escasso o material a respeito de como migrar as soluções existentes de computação convencional para a Computação nas Nuvens.

Nesse trabalho realizamos estudos a respeito de Computação nas Nuvens, Java e Google App Engine, afim de tentar compreender quais os problemas existentes em migrar aplicações Web em Java para o Google App Engine. Durante o estudo, foi realizado a migração de uma aplicação e através desse estudo foi levantado uma série de problemas e soluções.

O estudo concluiu que o Google App Engine é uma opção viável a migração e desenvolvimento de aplicações web nos moldes da utilizada no estudo de caso, ou seja,a aplicações com troca de mensagens, CRUD, processamento Batch e Ajax. Por conta de suas cotas gratuitas, ele é uma boa solução para aplicações pequenas. Entretanto, para aplicações mais complexas e que fazem uso de muitas APIs o trabalho de configuração acaba sendo muito tedioso e pouco usual, sendo necessário o uso *hackeamentos* muitas vezes. Esses problemas que foram resolvidos com *hackeamentos,* não constavam no sítio oficial e exigam um bom tempo de pesquisa. Para aplicações pequenas é possível prever os problemas trazidos por esses *hackeamentos*, mas a medida que aumentamos o número de APIs se torna mais complicado de acharmos os erros.

Outro problema é a migração para o Google Datastore. Esse banco utiliza uma tecnologia proprietária do Google e apresenta uma série de limitações que forçam mudança de código, e mudanças na estrutura do banco, pois alguns relacionamentos não são suportados. Existe a opção de utilizar o Google Cloud SQL para solucionatr esses problemas, mas o Google Cloud SQL é pago.

O Google App Engine também não apresenta um desempenho bruto elevado. Ele é bom em quesitos como escalabilidade, mas não desempenho. Além disso, depois de um tempo de ociosidade a aplicação pode ser migrada de uma VM para outra, o que aumenta o

tempo de resposta na primeira requisição.

Diante desses problemas, ficou claro que a aplicação ideal para migração são as aplicações mais simples, que não usem tantas APIs e que não precissem de processamento elevado. A migração de aplicações complexas provavelmente apresentará uma alta taxa de refatoração e de conflitos entre APIs.

REFERÊNCIAS BIBLIOGRÁFICAS

TAURION, Cezar. **Computacao em Nuvem**: Transformando o mundo da tecnologia. Rio de Janeiro, Brasport, 2009.

CELLARY, Wojciech; STRYKOWSKI, Sergiusz**. E-Government Based on Cloud Computing and Service-Oriented Architecture**. ICEGOV: the 3rd international conference on Theory and practice of electronic governance , Poznan, p.5-10, 10 nov. 2009. Disponível em:
< http://dl.acm.org/citation.cfm?id=1693045 >. Acesso em: 28 nov. 2011.

CHIRIGATI, Fernando Seabra. **Computação em Nuvem**. Rio de Janeiro, RJ. 2009. Disponível em:
< http://www.gta.ufrj.br/ensino/eel879/trabalhos_vf_2009_2/seabra/ >. Acesso em 28 nov. 2011.

ARMBRUST, Michael; FOX, Armando; GRIFFITH, Rean; JOSEPH, Anthony D. ; KATZ, Randy; KONWINSKI, Andy; LEE, Gunho; PATTERSON, David; RABKIN, Ariel; STOICA, Ion; ZAHARIA, Matei. **Above the clouds**: A berkeley view of cloud computing, 2009. Disponível em:
< http://x-integrate.de/x-in-cms.nsf/id/DE_Von_Regenmachern_und_Wolkenbruechen_-_Impact_2009_Nachlese/$file/abovetheclouds.pdf >. Acesso em: 28 nov. 2011.

MELL, Peter; GRANCE, Timothy. **The NIST Definition of Cloud Computing (Draft)**. Jan. 2011. Disponível em: < http://docs.ismgcorp.com/files/external/Draft-SP-800-145_cloud-definition.pdf >. Acesso em 28 nov. 2011.

VAQUERO, Luis M. et al. **A break in the clouds**: towards a cloud definition. Acm Sigcomm: Computer Communication Review, New York, Ny, Usa, n.39 , p.50-55, jan. 2009.

SUN MICROSYSTEMS. **Take Your Business to a Higher Level**: Sun Cloud-Computing Technology Scales Your Infrastructure To Take Advantage Of New Business Opportunities. Disponível em: < http://www.progression.com/casestudies/studies/Sun_Cloud_Computing.pdf >. Acesso em: 28 nov. 2011.

SILVA, FABRÍCIO. **Um estudo sobre os benefícios e os riscos de segurança na utilização de Cloud Computing**. 2010. 15 f. Artigo (Bacharel) - Unisuam, Rio de Janeiro, 2010. Disponível em: < http://fabriciorhs.files.wordpress.com/2011/03/cloud_computing.pdf >. Acesso em: 28 nov. 2011.

NOGUEIRA, Matheus Cadori; PEZZI, Daniel da Cunha. **A computação agora é nas nuvens**. Disponível em: < http://www.inst-informatica.pt/servicos/informacao-e-

documentacao/dossiers-tematicos/teste-dossier-tematico-no-7-cloud-computing/tendencias/a-computacao-agora-e-nas-nuvens/at_download/file >. Acesso em: 28 nov. 2011.

YOUSEFF, Lamia; BUTRICO, Maria; SILVA, Dilma. **Toward a Unified Ontology of Cloud Computing**, 2009 . Disponível em: < http://www.cs.ucsb.edu/~lyouseff/CCOntology/CloudOntology.pdf >. Acesso em: 29 nov. 2011.

BUYYA, Rajkumar. **Market-Oriented Cloud Computing**: Vision, Hype, and Reality for Delivering IT Services as Computing Utilities, 2008. Disponível em: < http://www.buyya.com/papers/hpcc2008_keynote_cloudcomputing.pdf >. Acesso em: 29 nov. 2011.

ZAHARIEV, Alexander. **Google App Engine**, 2009. Disponível em:

< http://www.cse.tkk.fi/en/publications/B/5/papers/1Zahariev_final.pdf >. Acesso em: 29 nov. 2011.

SOUSA, Flávio; MACHADO, Leonardo, 2010. **Computação em Nuvem**: Conceitos, Tecnologias, Aplicações e Desafios. Disponível em:< http://www.es.ufc.br/~flavio/files/Computacao_Nuvem.pdf >. Acesso em 29 nov. 2011.

VELTE, Anthony T.;VELTE, Toby J.;ELSENPETER, Robert. **Cloud Computing** : A pratical approach. 2010.

SANDERSON, DAN. **Programming Google App Engine**. 2010.

GOOGLE App Engine: Java Runtime Environment. Disponível em: < https://developers.google.com/appengine/docs/java/ >. Acesso em: 12 jun. 2013.

GOOGLE Cloud Storage: Overview. Disponível em: < https://developers.google.com/storage/docs/overview >. Acesso em: 12 jun. 2013.

JENDROCK, Eric et al. **The Java EE 6 Tutorial.** Disponível em: < http://docs.oracle.com/javaee/6/tutorial/doc/ >. Acesso em: 10 jun. 2013.

TRILHA do Aprendizado do Java EE e Java Web Disponível em: < https://netbeans.org/kb/trails/java-ee_pt_BR.html >. Acesso em: 12 jun. 2013.

MUKHAR, Kevin et al. Beginning Java EE 5: From Novice to Professional. New York,: Apress, 2005.

PITANGA, Talita. JavaServer Faces: A mais nova tecnologia Java para desenvolvimento WEB . Disponível em: < http://www.guj.com.br/content/articles/jsf/jsf.pdf >. Acesso em: 24 abr. 2013.

PIRES, Paulo. Fundamentos do JNDI. Rio de Janeiro: Universidade Federal do Estado do Rio de Janeiro, 2003. Disponível em: < http://www.uniriotec.br/~paulo.pires/cursos/TABD1/JNDI.pdf >.

PEREIRA, Austeclynio. Java Message Service (JMS). Conceito, Rio de Janeiro, 2005. Disponível em: < http://www.nce.ufrj.br/conceito/artigos/2005/012p1-2.htm >. Acesso em: set/out. 2005.

LIMA, Jean Carlos Rosário. WEB SERVICES (SOAP X REST). 2012. 41 f. Monografia (Tecnólogo) - Fatecsp, São Paulo, 2012.

YUAN, Michael J.. Comparação do PaaS em Java: Uma comparação técnica entre Google App Engine, Amazon Elastic Beanstalk e CloudBees RUN@Cloud. Disponível em: < http://www.ibm.com/developerworks/br/library/j-paasshootout/ >. Acesso em: 12 maio 2011.

SEAN, Sulivan. Advanced DAO programming: Learn techniques for building better DAOs. Disponível em: < http://www.ibm.com/developerworks/library/j-dao/ >. Acesso em: 07 out. 2003.

SILVA, Douglas Jean Dionizio da. Desenvolvimento de softwares utilizando injeção de dependência. 2012. 37 f. Relatorio Tecnico Científico (Graduando) - Faculdade de Tecnologia de Taquaritinga, Taquaritinga, 2012.

SILVA, Douglas M. Dornelles da; MACHADO, Guilherme Bertoni. **Serviço para execução de processamento em lotes (batch) com agendamento e gerenciamento pela web.** Disponível em: < http://periodicos.unesc.net/index.php/sulcomp/article/view/230 >. Acesso em: 12 jun. 2013.

APÊNDICE A – DESCRIÇÃO FUNCIONAL

I. Login

Ao entrar na aplicação o usuário encontra uma página com um formulário que tem os campos: nome e senha. Ao clicar no botão login, há uma validação do usuário e caso o usuário e senha estejam corretos, a página é redirecionada para uma outra página onde é carregado as funcionalidades que o usuário tem permissão. As funcionalidade sao:cadastro e lista. Na figura 1 há uma imagem da página.

Figura 1 – Página de Login da aplicação

II. Cadastro

Tendo feito o login e tendo acesso a funcionalidade de cadastro, o usuário vai ter acesso a uma página na qual há um link para o cadastro. Clicando nesse link é carregado um formulário com os campos:nome, sobrenome, senha, confirmação de senha, idade,sexo e funcionalidades.

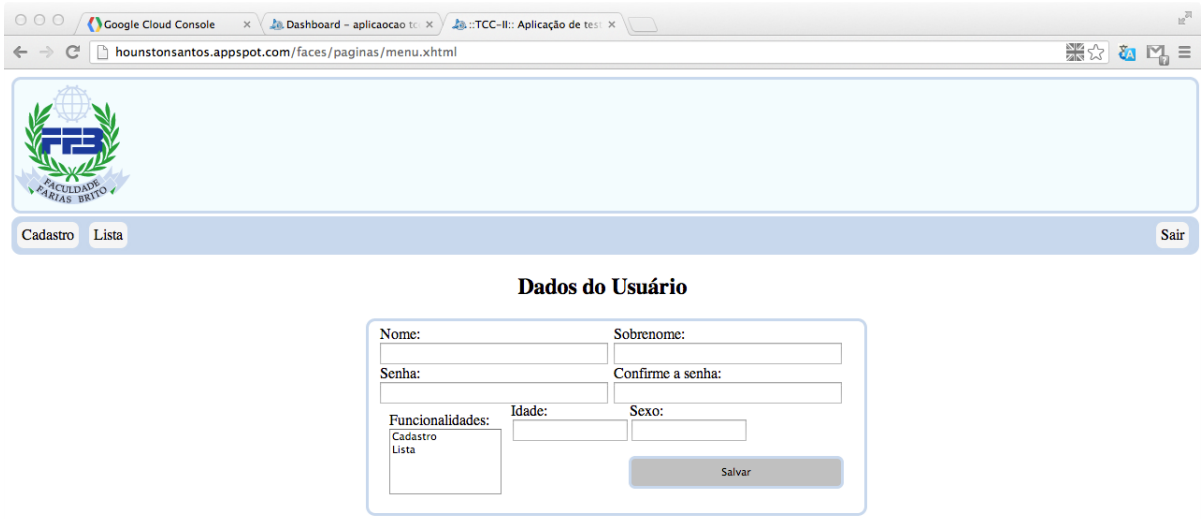

Figura 2 – Página de cadastro

III. Consulta, Atualização e Remoção

Para a consulta, atualização e remoção de usuário, existe uma única página. Para ter acesso a essa página o usuário deve realizar o login e clicar no link com o nome Lista.

Tendo clicado nesse link, o usuário é redirecionado para uma página onde é feita um consulta com todos os usuários. Além da lista com usuários, há um formulario de atualização. Para atualizar, o usuário deve procurar na lista o usuário que ele deseja atualizar, e então clicar no link atualizar dele. Feito isso, o formulario carrega os dados desse usuário e depois é só alterar os dados e clicar em salvar.

Para remover o usuáro, bastar clicar no link delete do usuário que se deseja remover.

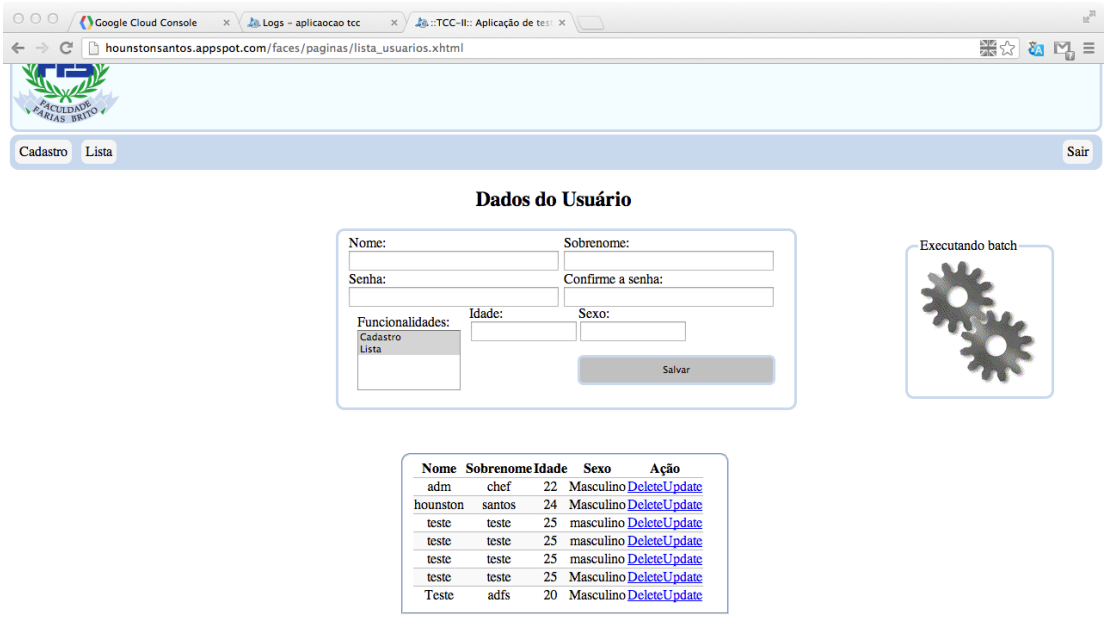

Figura 3 – Tela de consulta, atualização e remoção de usuário.

IV. Processo batch

Na tela Lista é possível ver uma engrenagem, isso pode ser verificado na Figura 3. Essa engrenagem aparece quando a aplicação executa um processo batch que migra os usuários de um banco para outro. Quando a aplicação não está executando esse processo, essa engrenagem desaparece.

V. Diagrama de caso de uso

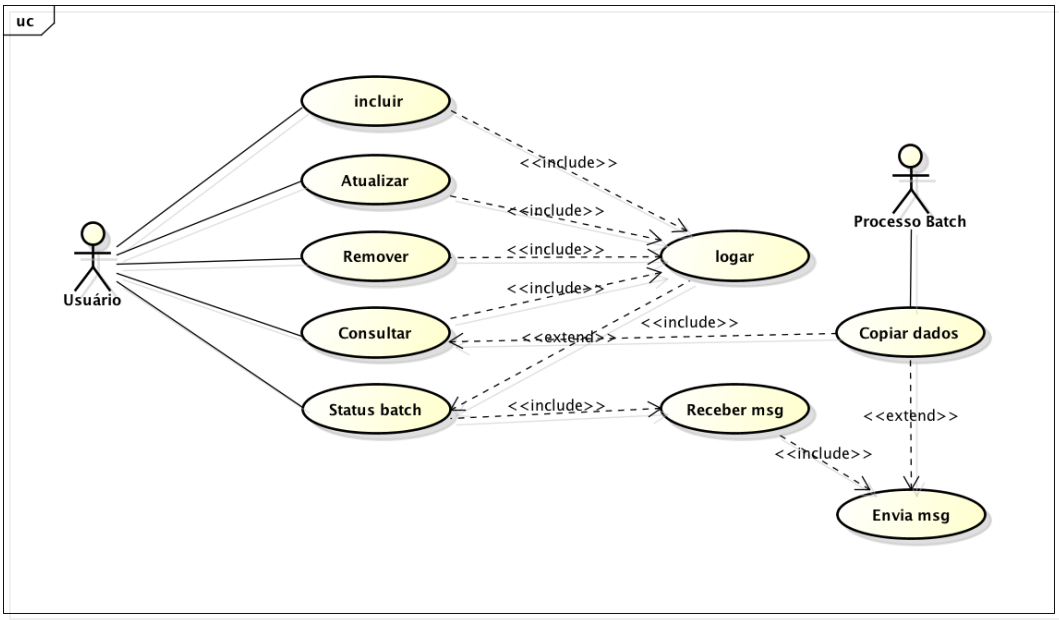

Figura 4 – Diagrama de caso de uso da aplicação de Teste para o TCC

APÊNDICE B – DIAGRAMA RELACIONAL

A estrutua do banco permanceu a mesma depois da migração para o Google Cloud SQL. Não é possível fazer um diagrama relacional para o banco Google Datastore, pois ele é um banco não-relacional. A figura 5 é a estrutura da aplicação.

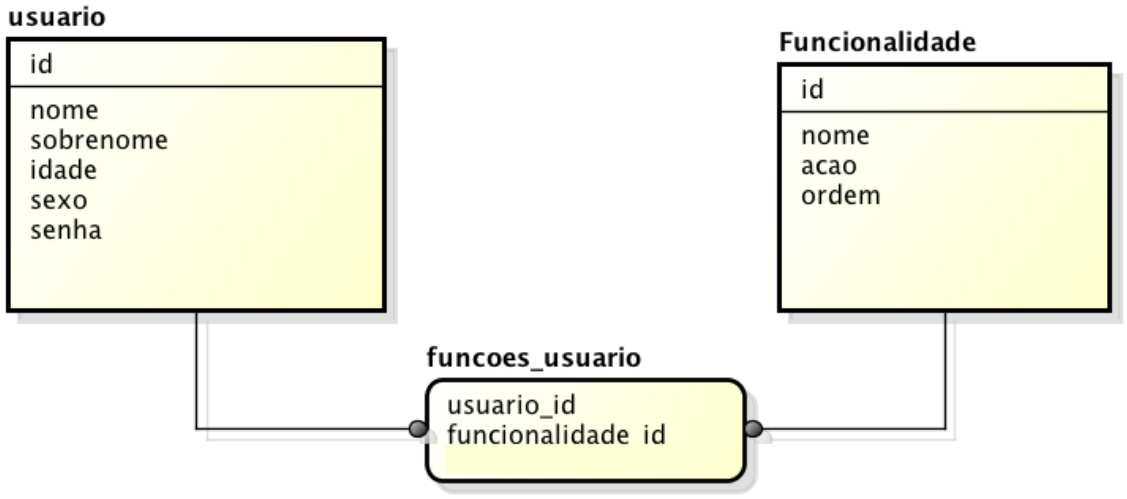

Figura 5 – Diagram estrutura relacional.