**CouchDB**

MIT em Big Data

Bloco C – Armazenamento Heterogêneo de Dados

Projeto de Bloco – CouchDB

06 de outubro de 2016

##### Ficha Técnica

|  |  |
| --- | --- |
| **Título** | CouchDB |
| **Curso** | MIT em Big Data |
| **Bloco C** | Armazenamento Heterogêneo de Dados |
| **Componentes do Grupo** | Herick Freitas  Leonardo Lima  Rodrigo Fernando Dias |
| **Coordenador** | Eduardo Morelli |

##### Glossário

**API** - é a sigla de *Application Programming Interface* que é um conjunto de rotinas e padrões de programação para acesso a um aplicativo de software ou plataforma baseado na web.

**Cluster** - é o nome dado a um sistema que relaciona dois ou mais computadores para que estes trabalhem de maneira conjunta no intuito de processar uma tarefa. Estas máquinas dividem entre si as atividades de processamento e executam as tarefas simultaneamente.

**HTTP** - *Hypertext Transfer Protocol* é um protocolo de comunicação utilizado para sistemas de informação de hipermedia distribuídos e colaborativos.

**JavaScript** - é uma linguagem de programação interpretada. Foi originalmente implementada como parte dos navegadores web para que *scripts* pudessem ser executados do lado do cliente e interagissem com o usuário sem a necessidade deste *script* passar pelo servidor, controlando o navegador, realizando comunicação assíncrona e alterando o conteúdo do documento exibido.

**JSON** - *JavaScript Object Notation* é uma formatação leve de troca de dados. Para seres humanos, é fácil de ler e escrever. Para máquinas, é fácil de interpretar e gerar.

**MapReduce** - é um modelo de programação desenhado para processar grandes volumes de dados em paralelo, dividindo o trabalho em um conjunto de tarefas independentes.

##### Sumário

[1. Introdução 6](#_Toc463490577)

[2. Detalhamento Teórico 9](#_Toc463490578)

[Banco de Dados Orientado a Documentos 9](#_Toc463490579)

[O Teorema CAP 10](#_Toc463490580)

[Replicação no CouchDB 11](#_Toc463490581)

[Particionamento no CouchDB 11](#_Toc463490582)

[Consistência Eventual 12](#_Toc463490583)

[B-trees 12](#_Toc463490584)

[Controle de Concorrência de Múltiplas Versões (MVCC) 12](#_Toc463490585)

[2.1. Arquitetura 13](#_Toc463490588)

[2.2. Instalação 14](#_Toc463490589)

[Dependências 15](#_Toc463490590)

[Download, Configuração e Instalação 16](#_Toc463490591)

[Considerações de Segurança 17](#_Toc463490592)

[Executando 17](#_Toc463490593)

[2.3. Interação 20](#_Toc463490594)

[Operações CRUD 20](#_Toc463490595)

[Criando Databases 20](#_Toc463490596)

[Criando Documentos 22](#_Toc463490597)

[Consultando Documentos 24](#_Toc463490598)

[Temporary Views 24](#_Toc463490599)

[Map 24](#_Toc463490600)

[Reduce 26](#_Toc463490601)

[Design Documents 27](#_Toc463490602)

[Querying Views 29](#_Toc463490603)

[Range 30](#_Toc463490604)

[Limit, Skip e Ordenar 30](#_Toc463490605)

[Group 31](#_Toc463490606)

[Include 33](#_Toc463490607)

[Atualizando Documentos 34](#_Toc463490608)

[Deletando Documentos 35](#_Toc463490609)

[Deletando Databases 35](#_Toc463490610)

[Otimização 36](#_Toc463490611)

[Algumas Dicas para Melhorar o Desempenho do Banco 36](#_Toc463490612)

[2.4. Replicação 37](#_Toc463490613)

[Banco de Dados *Replicator* 37](#_Toc463490614)

[2.5. Particionamento 39](#_Toc463490615)

[Configurando o Cluster 39](#_Toc463490616)

[2.6. Tarefas Administrativas 45](#_Toc463490617)

[Backup 45](#_Toc463490618)

[Segurança 46](#_Toc463490619)

[3. Conclusões 48](#_Toc463490620)

[4. Referências Bibliográficas 50](#_Toc463490621)

# Introdução

O modelo relacional, concebido na década de 70, vem sendo utilizado na maioria dos sistemas de gerenciamento de banco de dados. Este modelo apresenta algumas características importantes como a utilização de mecanismos para garantir a integridade dos dados, utilização da SQL (*Structured Query Language*) para manipulação e o procedimento de normalização.

Os mecanismos de garantia de integridade mais utilizados são as chaves primárias e estrangeiras. Aquela com o objetivo de garantir a unicidade dos registros de uma tabela, esta assegurando a dependência entre atributos em tabelas diferentes.

O Processo de normalização tem como objetivo a aplicação de regras em um projeto de banco de dados para garantir eficiência no acesso aos dados e a consistência no armazenamento.

Sistemas de Gerenciamento de Bancos de Dados (SGBDs) relacionais gerenciam o acesso, a manipulação e a organização dos dados. O SGBD oferece para os usuários a possibilidade de criar, consultar e alterar dados armazenados.

Com o elevado crescimento do volume de dados gerados por diversas fontes (empresas, dispositivos, redes sociais etc) e em diversos formatos (áudio, vídeo, texto etc) os SGBD relacionais começaram a apresentar limitações, principalmente quando se torna necessário escalar o próprio banco de dados, distribuindo-o por outras máquinas (escalabilidade horizontal). Tal abordagem é possível em um SGBD relacional, porém, não é realizada de maneira tão simples e possui custos elevados.

Neste contexto desenvolvedores começaram a pensar em uma abordagem alternativa de modelagem de bancos de dados. Surgem então as soluções não-relacionais que apresentam como principal proposta a redução ou até mesmo a eliminação da estruturação pouco flexível do modelo relacional.

No final da década de 90 surge o termo NoSQL[[1]](#footnote-1) a partir de um sistema relacional, mas que não apresentava interface SQL. Com o passar dos anos este termo passou a denominar soluções que promoviam uma alternativa ao modelo relacional. O propósito das soluções NoSQL não é substituir o modelo relacional como um todo, mas apenas em casos nos quais seja necessária uma maior flexibilidade da estruturação do banco com objetivo de alcançar performance adequada.

O presente trabalho tem como objetivo apresentar uma das soluções de banco de dados não-relacional. O CouchDB é um sistema de gerenciamento de bancos de dados não relacional, orientado a documentos que são armazenados no formato JSON.

De acordo com o último Ranking DB-Engines, o CouchDB está em 26º em popularidade. O Ranking classifica os sistemas de gerenciamentos de banco de dados a partir dos seguintes parâmetros: número de menções do sistema em sites; interesse geral no sistema; frequência de discussões técnicas sobre o sistema; relevância em redes sociais; entre outros. Por ser um banco ainda recente se comparado com a maioria dos melhores ranqueados, o CouchDB precisa ainda de algumas melhorias em sua tecnologia para que se torne mais atrativo e eficiente aos usuários. Entre os bancos orientados a documentos destacam-se o MongoDB (5ª posição) e o Couchbase (22ª posição) um pouco mais à frente do CouchDB.

Tabela – Opções de Consultas do CouchDB

|  |  |  |
| --- | --- | --- |
| Posição em Set/16 | SGBD | Modelo |
| 1º | Oracle | Relational DBMS |
| 2º | MySQL | Relational DBMS |
| 3º | Microsoft SQL Server | Relational DBMS |
| 4º | PostgreSQL | Relational DBMS |
| 5º | MongoDB | Document Store |
| 22º | Couchbase | Document Store |
| 26º | CouchDB | Document Store |

O presente estudo está dividido em quatro seções incluindo esta introdução. Na segunda seção é apresentado um detalhamento teórico do CouchDB onde são analisadas, detalhadamente, todas as ferramentas e configurações disponibilizadas pelo sistema. Durante o detalhamento são apresentados exemplos práticos da utilização dos recursos do banco (CRUD, replicação, *sharding* etc). A terceira seção são feitas algumas considerações conclusivas com relação ao sistema e na última seção são apresentados os materiais utilizados na construção deste estudo e que servem de referência para a utilização do CouchDB.

# Detalhamento Teórico

O projeto CouchDB, desenvolvido em linguagem C++, teve início em 2005 com Damien Katz. No ano de 2008 foi transferido para a plataforma Erlang OTP e passou a ser gerido pela Apache Software Foundation. O termo "*Couch*" é um acrônimo para "*Cluster Of Unreliable Commodity Hardware*", que reflete a meta do CouchDB de ser extremamente escalável, oferecendo alta disponibilidade e confiabilidade. O objetivo principal era tornar o CouchDB um banco de dados voltado para aplicações web, isto é, um banco de dados para a internet.

O Apache CouchDB é um sistema de banco de dados *open source*, orientado a documentos que armazena seus dados em arquivos JSON sem necessidade de esquema, tem o JavaScript como linguagem para manipulações, utiliza o conceito de arquitetura MapReduce (pensada para dar suporte à computação distribuída em larga escala) e acessível por intermédio de uma API RESTful. CouchDB suporta configurações *master-master* com detecção de conflitos automática e os dados do sistema possuem consistência eventual.

Atualmente o CouchDB é utilizado por empresas como a BBC, Meebo, Assay Depot, Engine Yard entre outros e também é parte integral de sistemas operacionais Linux Ubuntu.

#### Banco de Dados Orientado a Documentos

Os bancos orientados a documentos são uma das principais categorias de bancos NoSQL. Este tipo de banco de dados trabalha com coleções de documentos. Fazendo um paralelo com o modelo relacional, no lugar das tradicionais tabelas, utilizam-se grupamentos de dados denominados *collections*, que não possuem chaves (primária e/ou estrangeira) e nem transações. Dentro das *collections* são armazenados os documentos que no modelo relacional seriam os registros de uma tabela. Os documentos são representados como objetos de identificador único e como um conjunto de campos variados (*string*, *array*, lista, objetos etc).

A característica principal deste modelo é a não dependência de uma estrutura fixa para cada documento (livre de esquemas). Em um banco orientado a documentos podem ser realizadas atualizações na estrutura dos documentos sem ocasionar nenhum problema ao banco como um todo.

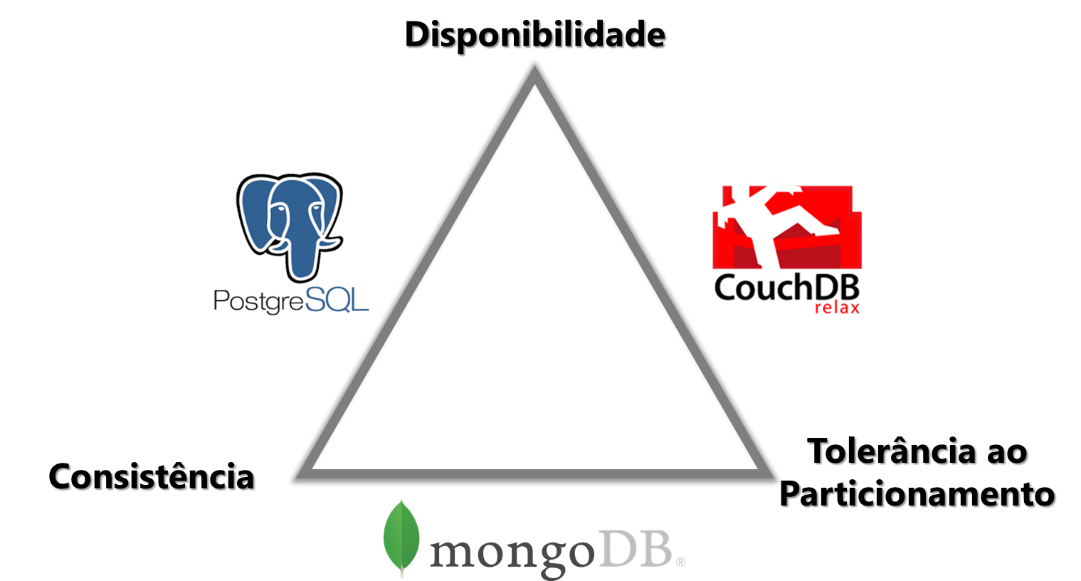
Conforme mencionado anteriormente, CouchDB utiliza o formato JSON (JavaScript Object Notation) para armazenar seus documentos.

#### O Teorema CAP

De acordo com teorema proposto por Brewer (2000), em sistemas computacionais distribuídos seria desejável obter consistência, disponibilidade e tolerância ao particionamento. Conforme provado, um sistema distribuído só pode garantir duas destas propriedades ao mesmo tempo. Este teorema ficou conhecido como teorema CAP (*Consistency*, *Availability* e *Partition* *Tolerance*).

Para garantir consistência, após uma transação o sistema deve manter a integridade das informações do banco, ou seja, todos os usuários devem ter acesso ao mesmo conjunto de dados. A disponibilidade significa que o sistema deve manter o acesso e as funcionalidades disponíveis para todos os usuários e a tolerância ao particionamento requer que os dados possam ser divididos por múltiplo servidores na rede.

Figura 1 – O Teorema CAP



CouchDB abre mão de uma consistência forte para garantir alta disponibilidade por meio do processo de replicação dos dados através de múltiplos servidores, isto é, podem ser realizadas atualizações em determinados nós de uma aplicação sem que seja necessário aguardar a comunicação entre os demais nós do cluster.

Com relação ao particionamento, como se trata de um banco de dados orientado a documentos de esquema flexível, possui alta tolerância a particionamento.

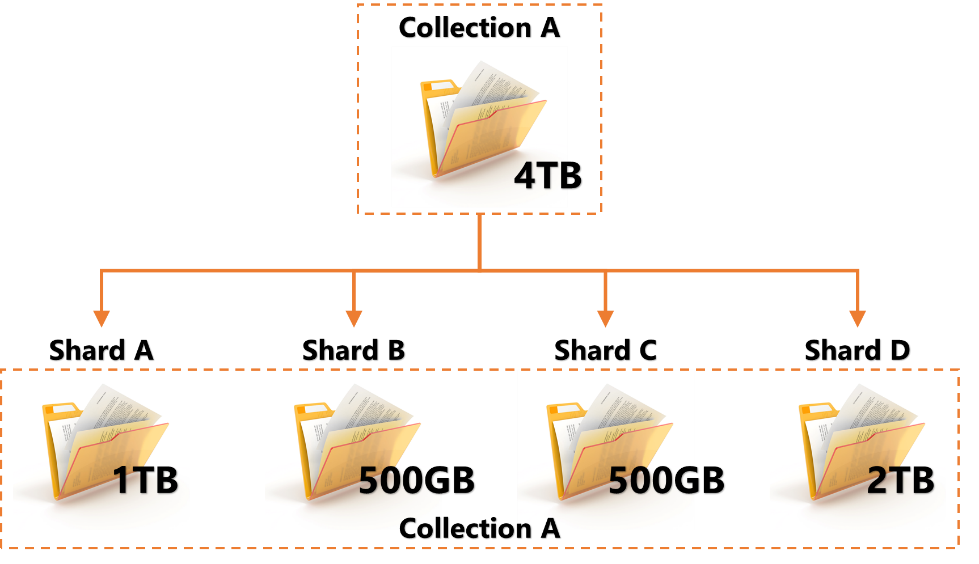
#### Replicação no CouchDB

A replicação funciona de forma incremental, ou seja, se durante a replicação alguma coisa der errado, como uma queda na conexão de rede, ele vai continuar de onde parou na próxima vez que for executado. Ele também só transfere os dados que forem necessários para sincronizar o banco de dados.

#### Particionamento no CouchDB

Dependendo do tipo de aplicação, algumas vezes, torna-se necessário o particionamento dos dados em múltiplos *shards* (fragmentos). Cada *shards* fica armazenado em um nó CouchDB e contém um subconjunto dos dados de uma *collection* (Figura 2).

Figura 2 – Sharding de uma Collection



É possível ter um ou mais *shards* de tamanhos variados em cada nó. Até a versão 1.6, o CouchDB não suportava nativamente esta forma de *clustering*. No entanto, existem ferramentas de terceiros que permitem a criação de particionamento de dados em *nodes* CouchDB. Essas ferramentas incluem BigCouch, Lounge e Pillow.

#### Consistência Eventual

O modelo de consistência eventual é caracterizado pelo pequeno número de ocorrências de atualizações que, quando ocorrem, não são imediatamente propagadas pelos nós. Este modelo, que é principalmente aplicado em bancos de dados onde prevalece requisições de leitura, necessita que eventualmente todas as atualizações sejam propagadas para todas as cópias.

Este sistema funciona de forma consistente para acessos a uma mesma réplica, porém quando múltiplas réplicas são utilizadas, em um curto período de tempo, podem ocorrer inconsistências.

#### B-trees

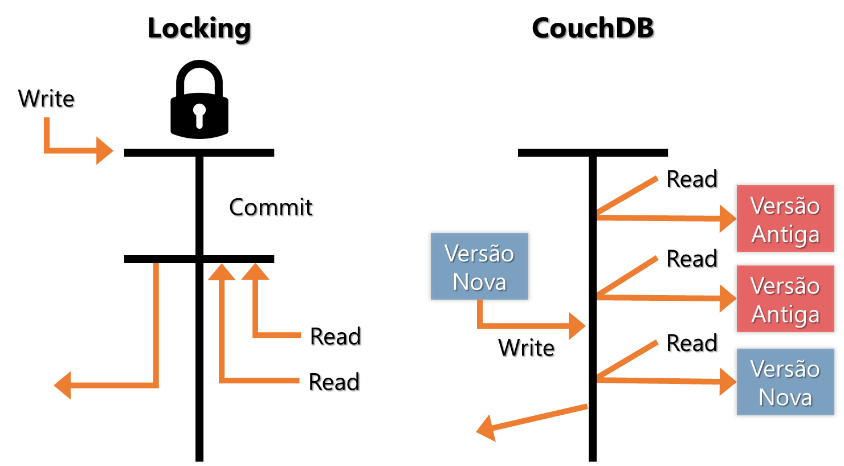
CouchDB utiliza uma estrutura de dados conhecida com *B-tree* para indexar seus documentos e *views*. *B-tree* é uma estrutura de dados ótima para armazenamento e recuperação rápida de grandes volumes de informações. Segundo Rudolf Bayer, idealizador da estrutura de dados B-tree, a utilização de *B-tree* garante um tempo de acesso inferior a dez ms mesmo para conjuntos de dados extremamente grandes.

Sua implementação no CouchDB é um pouco diferente. Mesmo mantendo as características principais, acrescenta o Controle de Concorrência de Múltiplas Versões.

#### Controle de Concorrência de Múltiplas Versões (MVCC)

O CouchDB utilizada uma estratégia chamada de Controle de Concorrência de Múltiplas Versões (MVCC). Neste caso, as requisições aos dados podem ser realizadas em paralelo. A ideia é criar múltiplas versões dos documentos e permitir a atualização sobre uma dessas versões, mantendo ainda a versão desatualizada. Assim, não há necessidade de se bloquear os itens de dados. A Figura 3 ilustra o contraste entre o sistema de *locking* dos SGBDs tradicionais e o MVCC do CouchDB.

Figura – Locking dos SGBDs Tradicionais e o MVCC do CouchDB





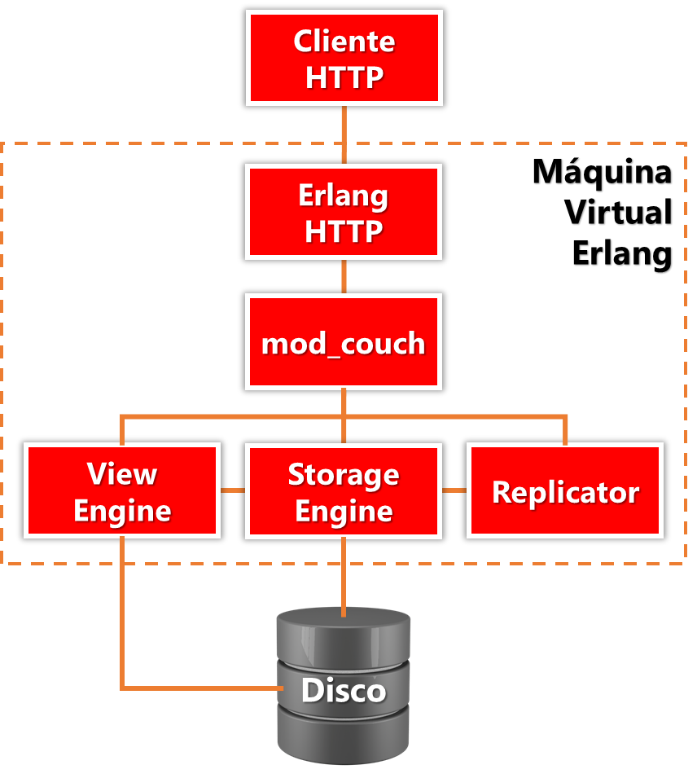
## Arquitetura

Além de contar com uma API RESTful para acesso e controle de dados, CouchDB também disponibiliza uma ferramenta de administração que possibilita gerenciar os bancos e documentos de forma individualizada. Esta ferramenta, chamada Futon até a versão 1.6 do CouchDB, pode ser acessada através do endereço http://localhost:5984/\_utils.

Um esquema com o modelo de arquitetura do CouchDB pode ser visto na Figura 4. Este modelo indica que a requisição feita por um cliente HTTP é recebida pelo módulo do CouchDB (***mod\_couch***) que faz o tratamento das solicitações. Observa-se também os processos que estão sendo executados no servidor e que são responsáveis por tarefas de replicação, visualização e armazenamento dos dados em disco.

A arquitetura do CouchDB é formada por um banco de dados *master* que possui um ou mais bancos de dados que são compostos por objetos JSON (documentos).

Figura – Arquitetura CouchDB



CouchDB possui suporte para replicação do banco de dados em diversos servidores com mecanismo para sincronização das informações. As replicações são realizadas de forma assíncrona, de forma que não compromete a velocidade de escrita e leitura (ANDERSON, 2010).

## Instalação

Nesta seção é mostrado todo o processo de instalação do CouchDB em sua versão 1.6.1. Foi utilizada neste processo uma máquina virtual (VM) com o sistema Linux CentOS 7 64-bit instalado[[2]](#footnote-2). Deve ficar claro que cada sistema tem suas particularidades. Além disso, com as atualizações dos arquivos de distribuição do CouchDB as dependências são frequentemente modificadas.

**# Preparando ambiente para instalação (modificando local para armazenar arquivos de instalação, criando variáveis etc).**

cd /tmp/

EPEL\_FILE=epel-release-latest-7.noarch.rpm

COUCH=apache-couchdb-1.6.1.tar.gz

rm -rf $EPEL\_FILE

#### Dependências

A instalação e o funcionamento do CouchDB requerem uma coleção de pacotes pré-instalados no sistema.

* Erlang OTP (>=R14B01, =<R17)
* ICU
* OpenSSL
* Mozilla SpiderMonkey (1.8.5)
* GNU Make
* GNU Compiler Collection
* libcurl
* help2man
* Python (>=2.7) for docs
* Python Sphinx (>=1.1.3)

É recomendada a instalação do Erlang OTP R13B-4 ou superior, sempre que possível. O pacote help2man só é preciso se desejar instalar o manual do CouchDB. Python e Sphinx só são necessários para a construção de documentação on-line.

**# Habilitando os repositórios EPEL e REMI para instalar a dependências do CouchDB.**

wget http://dl.fedoraproject.org/pub/epel/epel-release-latest-7.noarch.rpm

rpm -Uvh $EPEL\_FILE

**# Instalando as dependências do CouchDB.**

yum install autoconf autoconf-archive automake libtool perl-Test-Harness erlang libicu-devel js-devel curl-devel gcc-c++

#### Download, Configuração e Instalação

Depois de instaladas todas as dependências pode-se iniciar o processo de download e configuração do CouchDB.

**# Download e configuração do CouchDB.**

wget http://apache.mirror.digitalpacific.com.au/couchdb/source/1.6.1/apache-couchdb-1.6.1.tar.gz

tar -xzf $COUCH

cd apache-couchdb-1.6.1

./configure --with-erlang=/usr/lib64/erlang/usr/include

Se o processo for concluído com sucesso a seguinte mensagem será mostrada:

**You have configured Apache CouchDB, time to relax.**

Para concluir a instalação basta executar o *script* abaixo.

**# Instalando o CouchDB.**

make && make install

Se o processo for concluído com sucesso a seguinte mensagem será mostrada:

**You have installed Apache CouchDB, time to relax.**

#### Considerações de Segurança

Para iniciar a utilização do CouchDB algumas configurações são necessárias. Primeiro seria interessante criar um usuário especial para utilizar o banco de dados.

**# Adicionando o usuário couchdb.**

adduser -r --home /usr/local/var/lib/couchdb -M --shell /bin/bash --comment "CouchDB Administrator" couchdb

Alterar a propriedade dos diretórios do CouchDB executando o *script* abaixo.

**# Alterando o proprietário dos diretórios do CouchDB.**

chown -R couchdb:couchdb /usr/local/etc/couchdb

chown -R couchdb:couchdb /usr/local/var/lib/couchdb

chown -R couchdb:couchdb /usr/local/var/log/couchdb

chown -R couchdb:couchdb /usr/local/var/run/couchdb

Alterar a permissão dos diretórios do CouchDB executando o *script* abaixo.

**# Alterando a permissão dos diretórios do CouchDB.**

chmod 0770 /usr/local/etc/couchdb

chmod 0770 /usr/local/var/lib/couchdb

chmod 0770 /usr/local/var/log/couchdb

chmod 0770 /usr/local/var/run/couchdb

#### Executando

Configuração para iniciar o CouchDB (*as a daemon*) durante a inicialização do sistema.

**# Para iniciar durante o boot do sistema.**

ln -s /usr/local/etc/rc.d/couchdb /etc/init.d/couchdb

chkconfig --add couchdb

chkconfig --level 345 couchdb on

Configurações de firewall podem ser necessárias em alguns ambientes.

**# Obtendo uma lista de active zones.**

firewall-cmd --get-active-zones

**# Permissão**

firewall-cmd --zone=public --add-port=5984/tcp --permanent

**# Reiniciando o serviço de firewall.**

firewall-cmd --reload

Por padrão, o CouchDB pode ser acessado dentro do próprio servidor, se for acessá-lo a partir da web, é necessário modificar o arquivo local.ini e alterar as configurações da seção [httpd] conforme mostrado abaixo.

vi /usr/local/etc/couchdb/local.ini

[httpd]

port = 5984

bind\_address = 0.0.0.0

**#Pressione a Tecla ESC e digite...**

:wq

Iniciando o CouchDB manualmente com o usuário ***couchdb***. O parâmetro -b executa o serviço em segundo plano.

**# Iniciando o CouchDB manualmente.**

couchdb couchdb -b

Iniciando o CouchDB como um *daemon*.

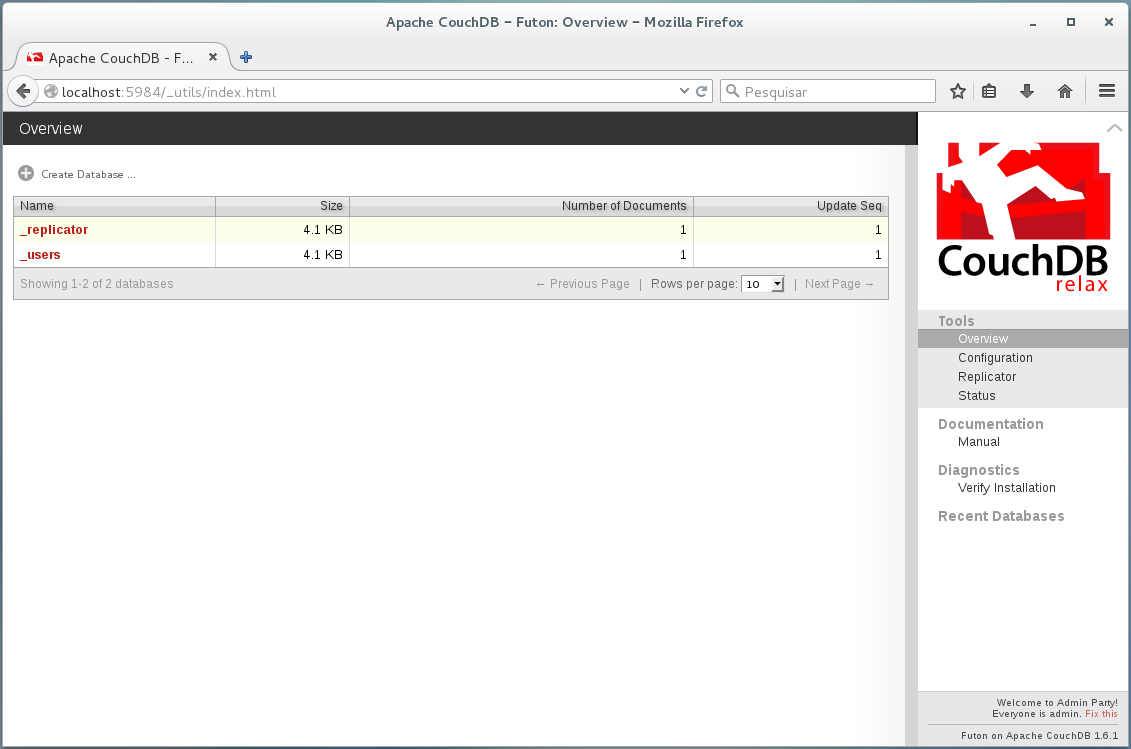
**# Iniciando o CouchDB as a daemon.**

/usr/local/etc/rc.d/couchdb start

Para controlar o CouchDB *daemon* basta utilizar o *script* acima com o seguinte conjunto de comandos: *start*, *stop*, *restart* e *status*.

Para verificar se tudo está funcionando corretamente basta acessar o seguinte endereço: http://127.0.0.1:5984/\_utils/index.html. A janela apresentada na Figura 5 deve aparecer com a página inicial do Futon.

Figura – Serviço CouchDB Executando (Futon)



## Interação

Nesta seção são apresentadas as ferramentas de administração e de acesso aos bancos de dados no CouchDB.

Para acessar os dados dos bancos de dados o CouchDB disponibiliza uma API REST. Essa API pode ser acessada por requisições HTTP *GET* e *POST* e retorna dados no formato de documentos JSON. Utilizando ferramentas de linha de comando (*curl* por exemplo) é possível fazer requisições *GET*, *POST*, *PUT* e *DELETE* e exibir as respostas HTTP recebidas do servidor da web.

Como ferramenta de administração o CouchDB possui um aplicativo web chamado Futon (Figura 5). Caso tenha instalado o CouchDB em sua máquina local na porta padrão 5984, é possível acessar o Futon apontando seu navegador para: http://localhost:5984/\_utils/.

### Operações CRUD

Para demonstrar o funcionamento do CouchDB e visualizar os resultados de consultas utilizando sua API, será utilizada a ferramenta de linha de comando *curl*. Este procedimento permite ilustrar as operações CRUD (*Create*, *Read*, *Update* e *Delete*) através de requisições e respostas HTTP (*GET*, *POST*, *PUT* e *DELETE*) obtidas do servidor CouchDB.

#### Criando Databases

Antes de criar documentos é necessário criar um database onde estes ficarão armazenados. O primeiro passo é verificar se o CouchDB está sendo executado.

**# Requisição**

**curl** http://localhost:5984

**# Resposta**

{

"couchdb":"Welcome",

"uuid":"d44520a609f34c7ff9fd73e9e29ff16d",

"version":"1.6.1",

"vendor":

{

"version":"1.6.1",

"name":"The Apache Software Foundation"

}

}

Para criar um novo banco de dados é preciso especificar a operação como sendo um *PUT*. Ao criar um banco de dados é necessário especificar o nome do mesmo na *URL* que está sendo enviada na solicitação. O exemplo abaixo cria o banco de dados **aluno**.

**# Requisição**

**curl -X PUT** http://localhost:5984/aluno

**# Resposta**

{"ok":true}

Caso seja enviada a mesma solicitação do exemplo anterior, uma mensagem de erro será exibida informando que não pode criar o banco de dados pois o arquivo já existe.

**# Requisição**

**curl -X PUT** http://localhost:5984/aluno

**# Resposta**

{

"error":"file\_exists",

"reason":"The database could not be created, the file already exists."

}

É possível recuperar as informações do banco de dados através de uma solicitação *GET* para a mesma *URL*.

**# Requisição**

**curl -X GET** http://localhost:5984/aluno

**# Resposta**

{

"db\_name":"aluno", "doc\_count":0,

"doc\_del\_count":0, "update\_seq":0,

"purge\_seq":0, "compact\_running":false,

"disk\_size":79, "data\_size":0,

"instance\_start\_time":"1474053474329959",

"disk\_format\_version":6, "committed\_update\_seq":0

}

#### Criando Documentos

Com o banco criado é possível criar documentos que serão armazenados neste. Para tanto, precisamos carregar o documento no corpo da requisição HTTP utilizando a opção ***-d***. Este procedimento adiciona o documento da requisição ao banco de dados.

**#Requisição**

**curl -H** 'Content-type: application/json' **-X PUT** http://localhost:5984/aluno/6254-9373-3347 **-d** \

'{

"\_id":"4277-8343-7320",

"nome":"Ludwing Van Beethoven",

"curso":"Marketing Digital",

"professores":[

"Raimundo",

"Girafales"

],

"local":"Petrocenter",

"inicio":"2016-06-04",

"horas":100

}'

**#Resposta**

{"ok":true,"id":"6254-9373-3347","rev":"1-90aee5e6e1c23ca147a53d0fde241d18"}

No CouchDB a requisição ***POST*** gera um novo documento no banco de dados e cria neste documento duas variáveis, um identificador (\_id) e um identificador de revisão (rev), ambos gerados pelo próprio sistema.

Para criar um documento com um id definido pelo usuário (exemplo acima) basta utilizar a solicitação ***PUT*** e especificar o nome do documento na *URL* de requisição. O nome do documento será utilizado como identificador.

Existem diversas formas de importar um conjunto de documentos (*collection*) para um banco de dados. Abaixo segue um exemplo de como realizar este procedimento.

**# A estrutura do arquivo JSON deve possuir o seguinte formato: {“docs” : [ ] }**

**# Observação: a variável \_id é opcional. Se não for definida o CouchDB atribui.**

**{**

**"docs": [**

{“\_id”: “0”, “nome”: “Aluno 1”, “idade”: 25},

**Documentos**

{“\_id”: “1”, “nome”: “Aluno 2”, “sexo”: “F”},

.

.

.

{“\_id”: “n”, “nome”: “Aluno n”, “idade”: 34, “sexo”: “M”}

**]**

**}**

**# Execute o seguinte comando para carregar os documentos no banco.**

**curl -d @**/home/***[username]***/Desktop/db.json **-H** "Content-type: application/json" **-X POST** http://localhost:5984/aluno/\_bulk\_docs

**#Resposta (quatro documentos carregados)**

[

{"ok":true,"id":"9590-6274-3666","rev":"1-dd044b3afc1a2d7fa21f7c0048aefb28"},

{"ok":true,"id":"8802-2651-7721","rev":"1-3e0c2d2a398a76b8fe91eb1d064591e5"},

{"ok":true,"id":"6338-5759-3182","rev":"1-5f9435ae042e4970bec063a798c3b00c"}

]

#### Consultando Documentos

O CouchDB disponibiliza uma série de formas de acesso aos dados em um banco. Pode-se gerar *Temporary Views* com funções de *Map* e *Reduce*, *Design* *Documents* para armazenar uma visualização e *Querying* *Views* onde são realizadas consultas dentro de visualizações predefinidas.

###### Temporary Views

*Temporary Views* são criadas a partir de funções de *Map* e *Reduce* escritas em linguagem JavaScript. Este tipo de visualização tende a ser mais lenta quando se trabalha com um número elevado de documentos. Para estes casos devem ser utilizadas visualizações salvas que serão apresentadas posteriormente neste trabalho (*Design Documents*).

###### Map

No processo de *Map*, os documentos são transformados (mapeados) de sua estrutura original para uma estrutura de chave/valor. Nos exemplos deste estudo será utilizado um banco de dados de alunos com informações de nomes dos alunos, cursos, professores, local, início das aulas e carga horária.

Considerando que no banco de dados **aluno** não existam nomes de alunos repetidos, utilizando uma função de *Map* que tenha como chave o atributo nome, cada documento será mapeado com exatamente um par de chave/valor (mapeamento um para um). O script abaixo mapeia o campo nome do banco aluno.

**#Função de *Map***

**function**(doc) {

**if** (doc.nome) {

**emit**(doc.nome);

}

}

**doc** – objeto JSON que representa um documento.

**emit()** – função que aceita dois argumentos (chave e valor) ambos opcionais.

A função acima verifica se o documento atual tem um nome de aluno e depois retorna o seu valor. Utilizando está função de *Map* foi gerada uma *Temporary View* conforme exemplo abaixo.

**#Requisição**

**curl -X** **POST** http://localhost:5984/aluno/\_temp\_view **-H** "Content-Type: application/json" **-d** '{"map": "**function**(doc) { **if** (doc.nome) { **emit**(doc.nome); } }" }'

**#Resposta**

{"total\_rows":4,"offset":0,"rows":[

{"id":"6338-5759-3182","key":"Albert Einstein","value":null},

{"id":"9590-6274-3666","key":"Charles Darwin","value":null},

{"id":"6254-9373-3347","key":"Ludwing Van Beethoven","value":null},

{"id":"8802-2651-7721","key":"Thomas Edison","value":null}

]}

Para exemplificar um tipo de mapeamento um-para-muitos pode-se construir a seguinte função de *Map* para o atributo professores.

**#Função de *Map***

**function**(doc) {

**if** (doc.professores) {

**for** (**var** i **in** doc. professores) {

**emit**(doc. professors[i]);

}

}

}

Gerando uma *Teporary View* utilizando a função de *Map* para professores.

**#Requisição**

**curl -X POST** http://localhost:5984/aluno/\_temp\_view **-H** "Content-Type: application/json" **-d** '{ "map": "**function**(doc) { **if** (doc.professores) { **for** (**var** i **in** doc.professores) { **emit**(doc.professores[i]); } } }"}'**#Resposta**

{"total\_rows":8,"offset":0,"rows":[

{"id":"6254-9373-3347","key":"Girafales","value":null},

{"id":"6338-5759-3182","key":"Girafales","value":null},

{"id":"8802-2651-7721","key":"Girafales","value":null},

{"id":"6254-9373-3347","key":"Raimundo","value":null},

{"id":"6338-5759-3182","key":"Raimundo","value":null},

{"id":"9590-6274-3666","key":"Raimundo","value":null},

{"id":"6338-5759-3182","key":"Sherman Klump","value":null},

{"id":"9590-6274-3666","key":"Sherman Klump","value":null}

]}

###### Reduce

Como visto no item anterior, a etapa de *Map* gera um conjunto de pares chave/valor que, opcionalmente, podem ser reduzidos a um único valor ou um grupo de valores. Esta etapa é conhecida como *Reduce*.

CouchDB tem três funções de *Reduce* já construídas, são elas: ***\_count***, ***\_sum*** e ***\_stats***. A função ***\_count*** retorna o número de valores mapeados na consulta, ***\_sum*** retorna o somatório destes valores e ***\_stats*** retorna um conjunto de estatísticas como soma, número de ocorrências, valores mínimo e máximo e soma dos quadrados. As funções ***\_sum*** e ***\_stats*** requerem que os valores envolvidos nos cálculos sejam numéricos.

O código abaixo cria três *Temporary Views* utilizando funções de *Map* e *Reduce* para exemplificar o funcionamento das três funções de *Reduce* internas do CouchDB.

**#Requisição utilizando funções de *Map* e *Reduce* e o parâmetro *group* que agrupa os resultados segunda a chave (*key*).**

**curl -X POST** http://localhost:5984/aluno/\_temp\_view?group=true **-H** "Content-Type: application/json" **-d** '{ "map": "**function**(doc) { **if** (doc.professores) { **for** (**var** i **in** doc.professores) { **emit**(doc.professores[i], doc.horas); } } }", "reduce": "**[FUNÇÃO DESEJADA]**"}'**#Resposta utilizando a função \_count.**

{"rows":[

{"key":"Girafales","value":3},

{"key":"Raimundo","value":3},

{"key":"Sherman Klump","value":2}

]}

**#Resposta utilizando a função \_sum.**

{"rows":[

{"key":"Girafales","value":690},

{"key":"Raimundo","value":1180},

{"key":"Sherman Klump","value":1080}

]}

**#Resposta utilizando a função \_stats.**

{"rows":[

{"key":"Girafales","value":{"sum":690,"count":3,"min":50,"max":540,"sumsqr":304100}},

{"key":"Raimundo","value":{"sum":1180,"count":3,"min":100,"max":540,"sumsqr":593200}},

{"key":"Sherman Klump","value":{"sum":1080,"count":2,"min":540,"max":540,

"sumsqr":583200}}

]}

###### Design Documents

As funções de *Map* e *Reduce* vistas nos itens anteriores geram *Temporary Views* mas também podem ser armazenadas permanentemente como uma *view* através de um *Design Document*.

Um *Design Document* fica armazenado no banco de dados juntamente com os demais documentos e pode conter diversas visualizações. Tem as mesmas características de um documento normal se diferenciando apenas pela funcionalidade e a formação de seu identificador (“\_id”: ”\_design/...).

**#Requisição para criar um *Design Document*, chamado consultas, para diversas *views***

**curl -X PUT** http://localhost:5984/aluno/\_design/consultas **-d** \

'{

"\_id": "\_design/consultas",

"language": "javascript",

"views": {

"nomes": {

"map": "**function**(doc) { **if** (doc.nome) { **emit**(doc.nome, doc.horas); } }",

"reduce": "\_stats"

},

"professores": {

"map": "**function**(doc) { **if** (doc.professores) { **for** (**var** i **in** doc.professores) {

**emit**(doc.professores[i], doc.horas); } } }",

"reduce": "\_stats"

},

"horas": {

"map": "**function**(doc) { **if** (doc.inicio) { **emit**(doc.inicio.**split**(\"-\"), doc.horas); } }",

"reduce": "\_stats"

}

} }'

**#Resposta**

{"ok":true,"id":"\_design/consultas","rev":"1-d1d6b3a37e99d01862e1b26d6db1bbea"}

Para consultar um *Design Document* basta utilizar o método *GET* do HTTP da seguinte forma:

**#Requisição para consultar um *Design Document***

**curl -X GET** http://localhost:5984/aluno/\_design/consultas/\_view/professores?group=true

**#Resposta**

{"rows":[

{"key":"Girafales","value":{"sum":690,"count":3,"min":50,"max":540,"sumsqr":304100}},

{"key":"Raimundo","value":{"sum":1180,"count":3,"min":100,"max":540,"sumsqr":593200}},

{"key":"Sherman Klump","value":{"sum":1080,"count":2,"min":540,"max":540,

"sumsqr":583200}}

]}

###### Querying Views

Com as visualizações criadas e armazenadas, pode-se gerar diversas consultas aos dados do banco. É possível limitar os resultados de uma consulta, agrupar, ordenar, definir se utiliza ou não as funções de *Reduce* etc.

A Tabela 2 mostra o conjunto de opções para configuração de consultas no CouchDB.

Tabela – Opções de Consultas do CouchDB

| Parâmetros | Descrição |
| --- | --- |
| *reduce* | Quando uma função *Reduce* está definida em um *Design Document*, está opção permite habilitar/desabilitar o uso desta função. |
| *startkey* | Determina o início de um intervalo |
| *startkey\_docid* | Determina em qual id do documento o intervalo deve começar. Deve ser utilizado em conjunto com *startkey*. |
| *endkey* | Determina o fim de um intervalo |
| *endkey\_docid* | Determina em qual id do documento o intervalo deve terminar. Deve ser utilizado em conjunto com *endkey*. |
| *inclusive\_end* | Indica se os valores definidos nos parâmetros *endkey* e *endkey\_docid* devem fazer parte do intervalo. |
| *key* | Filtra para um determinado valor dentro de uma chave. |
| *limit* | Número máximo de linhas incluídas em um resultado de consulta. |
| *skip* | Número de linhas que devem ser ignoradas de um resultado de consulta. |
| *descending* | Determina a ordenação dos resultados da consulta (crescente ou decrescente). |
| *group* | Permite agrupar os resultados de uma consulta. |
| *group\_level* | Quando a chave do documento é um *array* JSON, este parâmetro pode ser utilizado para determinar níveis diferentes de agrupamentos dos resultados. |
| *include\_docs* | Indica se todo o conteúdo dos documentos em uma consulta deve ser reportado. |
| *stale* | Este parâmetro permite que o CouchDB não atualize os resultados de uma *views*. |

Range

Especificando os parâmetros ***startkey*** e/ou ***endkey*** ou somente ***key*** é possível controlar pela chave (*key*) de uma *view* ou, opcionalmente, pelo id do documento (***startkey\_docid*** e/ou ***endkey\_docid***), as linhas resultantes de uma consulta.

**#Requisição**

**curl -X GET** http://localhost:5984/aluno/\_design/consultas/\_view/nomes **-G** \

**-d** **reduce**=false \

**-d startkey**='"l"' \

**-d endkey**='"z"'**#Resposta**

{"total\_rows":4,"offset":2,"rows":[

{"id":"6254-9373-3347","key":"Ludwing Van Beethoven","value":100},

{"id":"8802-2651-7721","key":"Thomas Edison","value":50}

]}

Limit, Skip e Ordenar

Em uma consulta se pode limitar o número de resultados retornados (***limit***), retornar os resultados a partir de um número determinado de ocorrências (***skip***) e colocar os resultados em ordem decrescente (***descending***).

**#Requisição**

**curl -X GET** http://localhost:5984/aluno/\_design/consultas/\_view/professores **-G** \

**-d** **reduce**=false \

**-d limit**=5

**#Resposta**

{"total\_rows":8,"offset":0,"rows":[

{"id":"6254-9373-3347","key":"Girafales","value":100},

{"id":"6338-5759-3182","key":"Girafales","value":540},

{"id":"8802-2651-7721","key":"Girafales","value":50},

{"id":"6254-9373-3347","key":"Raimundo","value":100},

{"id":"6338-5759-3182","key":"Raimundo","value":540}

]}

**#Requisição**

**curl -X GET** http://localhost:5984/aluno/\_design/consultas/\_view/professores **-G** \

**-d reduce**=false \

**-d limit**=5 \

**-d skip**=5

**#Resposta**

{"total\_rows":8,"offset":5,"rows":[

{"id":"9590-6274-3666","key":"Raimundo","value":540},

{"id":"6338-5759-3182","key":"Sherman Klump","value":540},

{"id":"9590-6274-3666","key":"Sherman Klump","value":540}

]}

**#Requisição**

**curl -X GET** http://localhost:5984/aluno/\_design/consultas/\_view/nomes **-G** \

**-d reduce**=false \

**-d descending**=true**#Resposta**

{"total\_rows":4,"offset":0,"rows":[

{"id":"8802-2651-7721","key":"Thomas Edison","value":50},

{"id":"6254-9373-3347","key":"Ludwing Van Beethoven","value":100},

{"id":"9590-6274-3666","key":"Charles Darwin","value":540},

{"id":"6338-5759-3182","key":"Albert Einstein","value":540}

]}

Group

É possível agrupar consultas por chaves ou por parte destas chaves. Para grupamentos utilizando todos os valores dentro de uma chave, esta pode possuir qualquer tipo de valor, mas para grupamentos utilizando partes de uma chave, os valores desta devem estar em *arrays* no arquivo JSON.

**#Requisição**

**curl -X GET** http://localhost:5984/aluno/\_design/consultas/\_view/professores **-G** \

**-d group**=true

**#Resposta**

{"rows":[

{"key":"Girafales","value":{"sum":690,"count":3,"min":50,"max":540,"sumsqr":304100}},

{"key":"Raimundo","value":{"sum":1180,"count":3,"min":100,"max":540,"sumsqr":593200}},

{"key":"Sherman Klump","value":{"sum":1080,"count":2,"min":540,"max":540,"sumsqr":583200}}

]}

Também pode-se agrupar os resultados em diferentes níveis de agrupamentos. Para tanto a chave do documento deve ser um *array* JSON. O código abaixo exemplifica diferentes níveis de grupamento dos resultados.

**#Requisição**

**curl -X GET** http://localhost:5984/aluno/\_design/consultas/\_view/horas **-G** \

**-d group**=true \

**-d group\_level**=3**#Resposta**

{"rows":[

{"key":["2014","01","01"],"value":{"sum":50,"count":1,"min":50,"max":50,"sumsqr":2500}},

{"key":["2016","01","04"],"value":{"sum":100,"count":1,"min":100,"max":100,"sumsqr":10000}},

{"key":["2016","01","05"],"value":{"sum":540,"count":1,"min":540,"max":540,"sumsqr":291600}},

{"key":["2016","10","10"],"value":{"sum":540,"count":1,"min":540,"max":540,"sumsqr":291600}}

]}

**#Requisição**

**curl -X GET** http://localhost:5984/aluno/\_design/consultas/\_view/horas **-G** \

**-d group**=true \

**-d group\_level**=2

**#Resposta**

{"rows":[

{"key":["2014","01"],"value":{"sum":50,"count":1,"min":50,"max":50,"sumsqr":2500}},

{"key":["2016","01"],"value":{"sum":640,"count":2,"min":100,"max":540,"sumsqr":301600}},

{"key":["2016","10"],"value":{"sum":540,"count":1,"min":540,"max":540,"sumsqr":291600}}

]}

**#Requisição**

**curl -X GET** http://localhost:5984/aluno/\_design/consultas/\_view/horas **-G** \

**-d group**=true \

**-d group\_level**=1

**#Resposta**

{"rows":[

{"key":["2014"],"value":{"sum":50,"count":1,"min":50,"max":50,"sumsqr":2500}},

{"key":["2016"],"value":{"sum":1180,"count":3,"min":100,"max":540,"sumsqr":593200}}

]}

Include

A opção ***include\_docs*** permite incluir no resultado de uma consulta todo o conteúdo dos documentos resultantes.

**#Requisição**

**curl -X GET** http://localhost:5984/aluno/\_design/consultas/\_view/nomes **-G** \

**-d reduce**=false \

**-d include\_docs**=true

**#Resposta**

{"total\_rows":4,"offset":0,"rows":[

{"id":"6338-5759-3182","key":"Albert Einstein","value":540,"doc":{"\_id":"6338-5759-

3182","\_rev":"1-5f9435ae042e4970bec063a798c3b00c","nome":"Albert Einstein",

"curso":"MIT em Big Data","professores":["Sherman Klump","Raimundo","Girafales"],

"local":"Infnet", "inicio":"2016-10-10","horas":540}},

{"id":"9590-6274-3666","key":"Charles Darwin","value":540,"doc":{"\_id":"9590-6274-

3666","\_rev":"1-dd044b3afc1a2d7fa21f7c0048aefb28","nome":"Charles Darwin",

"curso":"MIT em Big Data","professores":["Sherman Klump","Raimundo"], "local":"Infnet",

"inicio":"2016-01-05","horas":540}},

{"id":"6254-9373-3347","key":"Ludwing Van Beethoven","value":100,"doc":{"\_id":"6254-

9373-3347","\_rev":"1-90aee5e6e1c23ca147a53d0fde241d18","nome":"Ludwing Van

Beethoven","curso":"Marketing Digital", "professores":["Raimundo","Girafales"],

"local":"Petrocenter", "inicio":"2016-01-04","horas":100}},

{"id":"8802-2651-7721","key":"Thomas Edison","value":50,"doc":{"\_id":"8802-2651-

7721","\_rev":"1-3e0c2d2a398a76b8fe91eb1d064591e5","nome":"Thomas

Edison","curso":"Desenvolvedor Java", "professores":["Girafales"], "local":"Infnet",

"inicio":"2014-01-01", "horas":50}}

]}

#### Atualizando Documentos

Para atualizar um documento utiliza-se a requisição *PUT* juntamente os novos dados do documento, o id e o id de revisão. O id de revisão é utilizado como mecanismo de segurança pois não é possível atualizar um documento sem conhecer sua versão atual.

**#Requisição**

**curl -H** 'Content-type: application/json' **-X PUT** http://localhost:5984/aluno/6254-9373-3347 **-d** \

'{

"\_id":"6254-9373-3347",

"\_rev":"1-90aee5e6e1c23ca147a53d0fde241d18",

"nome":"Ludwing Van Beethoven",

"curso":"Data Mining",

"professores":[

"Raimundo",

"Girafales",

"Pardal"

],

"local":"ENCE",

"inicio":"2016-01-04",

"horas":100

}'

**#Resposta**

{"ok":true,"id":"6254-9373-3347","rev":"2-d117809ad39747a79ba55d49f415ebfd"}

No CouchDB quando atualizamos um documento fazemos a modificação de todo o seu conteúdo, ou seja, não se pode fazer modificações pontuais em um documento (adicionar um campo, modificar explicitamente apenas um campo do documento etc). O que o CouchDB faz em um processo de atualização é escrever uma versão totalmente nova do documento utilizando o mesmo identificador (id).

#### Deletando Documentos

Para deletar um documento de um banco do CouchDB utilizamos o comando HTTP *DELETE*. Igualmente ao processo de atualização, precisamos fornecer o id e o id de revisão do documento.

**#Requisição**

**curl -H** 'Content-type: application/json' **-X DELETE** http://localhost:5984/aluno/6254-9373-3347?rev=2-d117809ad39747a79ba55d49f415ebfd

**#Resposta**

{"ok":true,"id":"6254-9373-3347","rev":"3-fa824e646dbdd5f6ee5f81825ea197e2"}

#### Deletando Databases

Para finalizar, o *script* abaixo exemplifica a forma de excluir um banco de dados no CouchDB. Este procedimento não solicita nenhum tipo confirmação.

**# Requisição**

**curl -X DELETE** http://localhost:5984/aluno

**#Resposta**

{"ok":true}

### Otimização

Com o crescimento do volume de dados no banco, consultas tendem a ficar mais lentas. Para tanto existem algumas medidas que podem ajudar a melhorar o desempenho de um banco de dados no CouchDB. Existem diversas opções disponíveis quando se trata de tuning no CouchDB, mas cada situação deve ser avaliada individualmente.

O CouchDB é um sistema gerenciador de banco de dados com esquema extremamente flexível na concepção de documentos para seus dados. Entretanto, as decisões sobre como projetar seus documentos podem ter ação extremamente impactante no desempenho e na escalabilidade do banco.

#### Algumas Dicas para Melhorar o Desempenho do Banco

* Evite utilizar IDs aleatórios para os documentos. CouchDB terá melhor performance quando as consultas forem em documentos com IDs sequencias;
* Configure o parâmetro ***include\_docs*** como *false* quando consultar views;
* Utilize algum sistema de *caching* para otimizar a alocação de memória do sistema e otimizar as consultas. Sistemas de *caching* permitem gerenciar a memória do sistema de forma mais eficiente, diminuindo a quantidade alocada para tarefas onde se tem mais do que precisa e transferindo para tarefas onde se tem menos;
* Sempre que possível utilize a API para realizar carga de documentos em blocos. Funciona como um mecanismo de importação de diversos documentos para o banco de dados. Este procedimento pode ser visto na seção **Criando Documentos**.
* Durante operações de escrita no banco habilite o modo *batch* (utilize **?batch=ok** após a *URL* em requisições *POST* e *PUT*). Com este modo habilitado o CouchDB irá salvar os documentos em memória e depois descarregá-los em lotes para disco;
* Configure a opção ***delayed\_commits*** como *true* na seção de configurações do CouchDB. Esta opção impede que o CouchDB realize *buffer* de informações em disco após cada operação de escrita no banco;
* Execute a compactação de bancos de dados e *views* e também a limpeza de *views* em período onde a aplicação esteja sendo menos solicitada;
* Distribua as *views* por múltiplos *Design Documents*. Isto deve acelerar as operações de limpeza e compactação das *views*;
* Invista em hardware de alta performance. Discos rápidos, CPUs de boa performance com múltiplos núcleos e bastante memória são indispensáveis para um bom desempenho do CouchDB.

## Replicação

A replicação no CouchDB é um processo incremental, podendo ser unidirecional (*master-slave*) ou bidirecional[[3]](#footnote-3) (*master-master*), que envolve duas bases de dados trabalhando no esquema *source*/*target*.

O objetivo da replicação é garantir que, no final de um processo, todos os documentos (criados, alterados e excluídos) do banco de dados *source* estejam também no banco de dados *target*. Este processo possibilita fornecer ao usuário acesso aos dados com baixa latência, não importando sua localização. Os bancos de dados envolvidos na replicação podem estar no mesmo servidor ou em servidores diferentes.

#### Banco de Dados *Replicator*

O banco de dados ***\_replicator*** funciona como qualquer outro no CouchDB, mas os documentos adicionados neste banco contêm parâmetros de configuração para replicações. As replicações podem ser configuradas através do Futon, de um console de administração web ou por requisições HTTP ao banco ***\_replicator***.

O *script* abaixo cria um documento (*replica\_a\_to\_b*) com as configurações de uma réplica do banco de dados **aluno\_a** (*source*), cria o banco de dados de destino (*create\_target = true*) e o chama de **aluno\_b** (*target*). Determina também que o processo de replicação deverá ser contínuo.

**#Requisição**

**curl -X POST** http://localhost:5984/\_replicator **-H** "Content-Type: application/json" **-d** \

'{

"\_id":"replica\_a\_to\_b",

"source":"http://localhost:5984/aluno\_a",

"target":"http://localhost:5984/aluno\_b",

"create\_target":true,

"continuous":true

}'

**#Resposta**

{"ok":true,"id":"replica\_a\_to\_b","rev":"1-af1e62ba0f66cde061c58bfa02f38d95"}

A replicação do exemplo segue o modelo *master-slave*, ou seja, é unidirecional. Para configurar uma replicação bidirecional (*master-master*) basta adicionar o seguinte documento ao banco ***\_replicator***:

**#Requisição**

**curl -X POST** http://localhost:5984/\_replicator **-H** "Content-Type: application/json" **-d** \

'{

"\_id":"replica\_b\_to\_a",

"source":"http://localhost:5984/aluno\_b",

"target":"http://localhost:5984/aluno\_a",

"continuous":true

}'

**#Resposta**

{"ok":true,"id":"replica\_b\_to\_a","rev":"1-45223b24edb29e06fdb85b2e6c22fb35"}

Para cancelar um processo de replicação basta excluir o documento que configura a réplica.

## Particionamento

Conforme apresentado na seção anterior, a replicação é de grande utilidade, mas pode não suprir totalmente as necessidades de escalabilidade da aplicação. Com o crescimento dos dados torna-se necessário particionar os bancos através de múltiplo *shards*. Cada *shard* conteria um subconjunto dos dados e ficaria armazenado em um nó do CouchDB, sendo possível ter um ou mais *shards* em um mesmo nó.

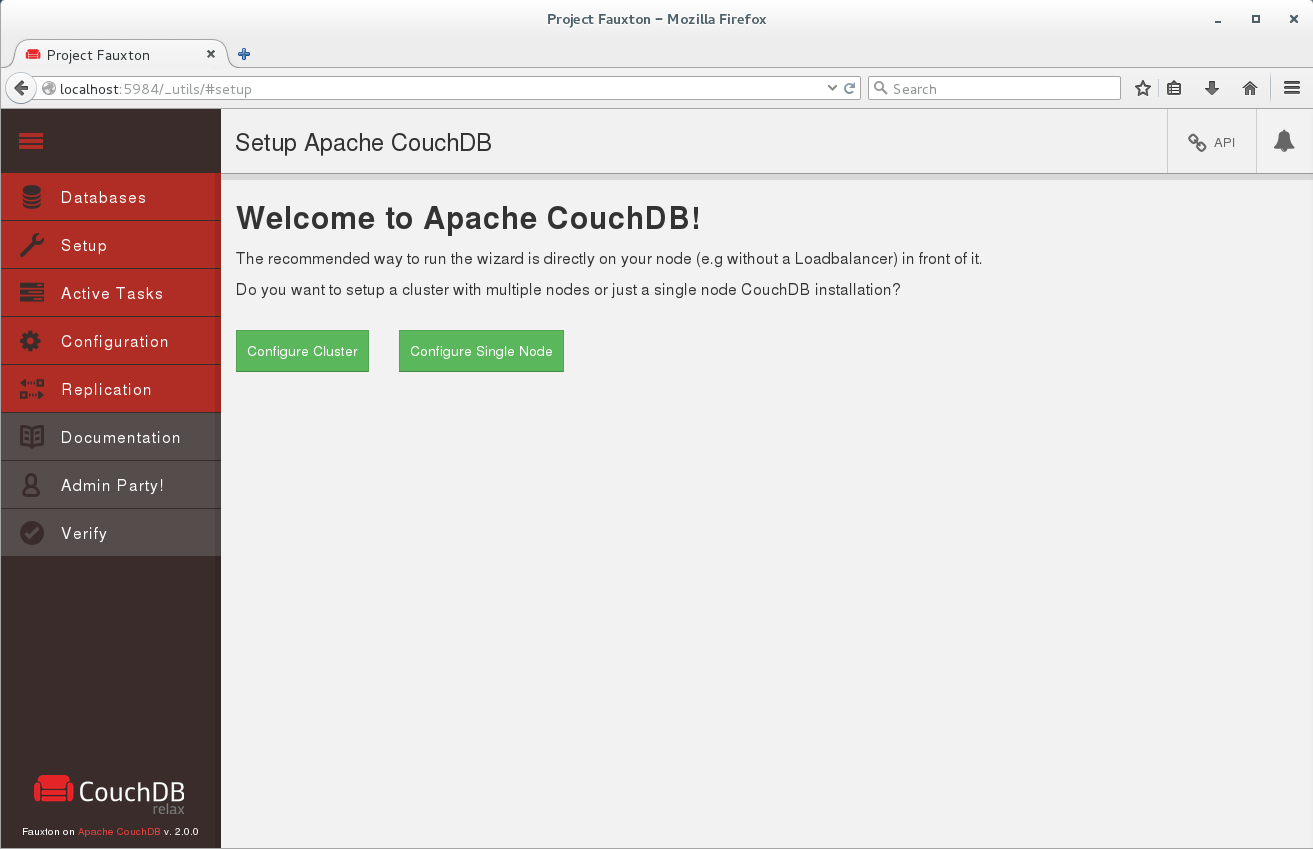
Até a versão 1.6.1, CouchDB não oferecia suporte para este tipo de particionamento sendo necessário utilizar ferramentas de terceiros para possibilitar a criação de particionamento dos dados de uma aplicação através de nós CouchDB. Algumas destas ferramentas incluem BigCouch, Lounge e Pillow.

Com o lançamento da versão 2.0, o Apache CouchDB começa a oferecer suporte nativo a este tipo de particionamento[[4]](#footnote-4).

#### Configurando o Cluster

Configurar corretamente um *cluster* em aplicação Erlang pode ser uma tarefa assustadora. Felizmente o CouchDB 2.0 oferece, através de sua ferramenta e administração (Fauxton), um assistente de configuração de *cluster* (Figura 6).

Figura – Cluster Setup Wizard (Fauxton)



Acesse http://localhost:5984/\_utils/#setup para iniciar o assistente de configuração. Será solicitado que escolha entre configurar um *cluster* ou um *single-node*. Iniciando a configuração do *cluster* será necessário fornecer as credenciais de administrador e o endereço IP da máquina. Para adicionar mais nós, execute este mesmo procedimento em outras máquinas.

Antes de adicionar os nós que irão formar o cluster é necessário torná-los visíveis publicamente e configurar um usuário administrador. O seguinte script deve ser executado em cada nó da aplicação.

**curl -X PUT** http://localhost:5984/\_node/couchdb@***[endereço IP do nó]***/\_config/admins/admin **-d** '"password"'

**curl -X PUT** http://localhost:5984/\_node/couchdb@***[endereço IP do nó]***/\_config/chttpd/bind\_address **-d** '"0.0.0.0"'

Após adicionar todos os nós basta clicar em *Configure Cluster* para finalizar o *setup*. Para visualizar uma listagem de todos os nós do cluster acesse http://localhost:5984/\_membership.

**#Requisição**

**curl -X GET** http://localhost:5984/\_membership **--user** ***[usuário administrador]***

**#Resposta**

{

"all\_nodes":[

"node1@localhost"

],

"cluster\_nodes":[

"node1@localhost"

]

}

**Adicionando um Nó**

Para adicionar um nó basta seguir o seguinte procedimento:

**curl -X PUT** http://localhost:5986/\_nodes/node2@localhostA **-d** {}

Acessando novamente http://localhost:5984/\_membership ou http://localhostA:5984 /\_membership obtém-se a seguinte resposta com o novo nó configurado.

{

"all\_nodes":[

"node1@localhost",

"node2@localhostA"

],

"cluster\_nodes":[

"node1@localhost",

"node2@localhostA"

]

}

**Removendo um Nó**

O procedimento de remoção de um nó do *cluster* é bastante simples mas deve ser executado com atenção para não ocorrer perda de informação. Antes de remover um nó do *cluster* deve-se assegurar que todos os *shards* foram retirados deste nó e arquivados em outro.

Para remover o node2 do servidor localhostA o seguinte *script* deve ser executado:

**curl -X DELETE** http://localhost:5986/\_nodes/node2@localhostA **-d** {}

**Gerenciando o Banco de Dados**

Para criar um banco de dados em um ambiente particionado é necessário fornecer o número de réplicas (n) e também o número de *shards* (q) em que o banco ficará dividido. O exemplo abaixo cria o banco aluno com três réplicas e quatro *shards*.

**curl -X PUT** http://localhost:5984/aluno?n=3&q=4 **--user** ***[usuário administrador]***

Para apagar um baco de dados o processo é semelhante ao já apresentado anteriormente.

**curl -X DELETE** http://localhost:5984/aluno **--user** ***[usuário administrador]***

**Redimensionamento do Cluster**

Em geral uma aplicação começa com um conjunto menor de informações e vai ganhando dimensão com o passar do tempo. Inicialmente pode ser necessário poucos nós, mas com o passar do tempo e o crescimento do banco, torna-se indispensável redimensionar o cluster.

Para exemplificar o procedimento considere um nó com um banco de dados (**aluno**) sem réplicas em com apenas dois *shards*.

**curl -X PUT** http://localhost:5984/aluno?n=1&q=2 **--user** ***[usuário administrador]***

Consultando o painel de controle e visualizando o banco de dados **\_dbs** é possível acessar os metadados de cada banco armazenado e editar os seus documentos. Acessando o documento **aluno** dentro do banco **\_dbs** teríamos, entre outros campos, as seguintes informações:

**#Requisição**

**curl -X GET** http://localhost:5986/\_dbs/aluno

**#Resposta**

{

…,

"changelog": [

["add",

"00000000-7fffffff",

"node1@localhost"],

["add",

"80000000-ffffffff",

"node1@localhost"]

],

"by\_node": {

"node1@localhost": [

"00000000-7fffffff",

"80000000-ffffffff"]

},

"by\_range": {

"00000000-7fffffff": [

"node1@localhost"],

"80000000-ffffffff": [

"node1@localhost"]

}

}

Utilize o Fauxton para editar os metadados do documento aluno da seguinte maneira:

{

…,

"changelog": [

["add",

"00000000-7fffffff",

"node1@localhost"],

["add",

"80000000-ffffffff",

"node1@localhost"],

["add",

"00000000-7fffffff",

"node2@localhostA"],

["add",

"80000000-ffffffff",

"node2@localhostA"]

],

"by\_node": {

"node1@localhost": [

"00000000-7fffffff",

"80000000-ffffffff"],

"node2@localhostA": [

"00000000-7fffffff",

"80000000-ffffffff"]

},

"by\_range": {

"00000000-7fffffff": [

"node1@localhost",

"node2@localhostA"],

"80000000-ffffffff": [

"node1@localhost",

"node2@localhostA"]

}

}

Salve as alterações para o *shard* ser replicado em node2.

## Tarefas Administrativas

Nas próximas seções são apresentadas algumas ferramentas de administração do CouchDB como procedimentos de backup e rotinas de segurança.

### Backup

CouchDB não possui uma ferramenta específica para geração de *backups*. Para isso deve-se utilizar o recurso de replicação (conforme apresentado na seção sobre replicação).

O *script* abaixo cria um documento (**backup\_aluno**) com as configurações de uma réplica/*backup* do banco de dados **aluno** (*source*), cria o banco de dados de destino (*create\_target = true*) e o chama de **backup\_aluno** (*target*). Determina também que o processo de replicação não deverá ser contínuo.

**#Requisição**

**curl -X POST** http://localhost:5984/\_replicator **-H** "Content-Type: application/json" **-d** \

'{

"\_id":"backup\_aluno",

"source":"http://localhost:5984/aluno",

"target":"http://localhost:5984/backup\_aluno",

"create\_target":true,

"continuous":false

}'

O banco de dados **backup\_aluno** é uma cópia exata de **aluno** e pode ser utilizado para restaurar os dados deste banco em caso de perda de informações.

### Segurança

Nesta seção serão apresentados alguns mecanismos básicos de segurança do CouchDB.

**Criando um Novo Usuário Administrador**

Através da API e utilizando *curl* é possível criar um novo usuário administrador da seguinte maneira.

**curl -X PUT** http://localhost:5984/\_config/admins/gisele **-d** '"password"'

O script acima cria um administrador chamado ***gisele*** com a senha ***password***.

**Autenticação Básica**

Depois de definido um administrador não será mais possível gerar modificação no sistema sem as credenciais corretas de administração.

**#Requisição**

**curl -X PUT** http://localhost:5984/aluno

**#Resposta**

{"error":"unauthorized","reason":"You are not a server admin."}

Tentando com as credenciais corretas da seguinte forma: ***[nome do usuário]:[senha]@***.

**#Requisição**

**curl -X PUT** http://**gisele**:**password**@localhost:5984/aluno

**#Resposta**

{"ok":true}

**Criando um Novo Usuário**

O código abaixo cria um usuário padrão com login ***rodrigo*** a senha ***password***.

**curl -X PUT** http://localhost:5984/\_users/org.couchdb.user:rodrigo **-H** "Accept: application/json" **-H** "Content-Type: application/json" **-d** '{"name": "rodrigo", "password": "password", "roles": [], "type": "user"}'

# Conclusões

O projeto Apache CouchDB está em constante desenvolvimento e conforme novas releases são disponibilizadas, este se torna uma opção cada vez mais completa e atraente para gerenciamento de bancos de dados orientados a documentos. A utilização do CouchDB vem se tornando cada vez mais comum em aplicativos web, aplicações para dispositivos móveis e como parte de aplicações de grandes empresas como o Facebook e a BBC.

Na BBC, por exemplo, o CouchDB foi escolhido para criar uma configuração multi-master que garanta a migração automática da aplicação para outros servidores em caso de falha em determinado nó do cluster. Isto lhes permite usar trinta e dois nós divididos entre dois datacenters. Dos dezesseis nós em cada datacenter, oito são nós primários e os outros oito são nós de backup. Esta solução lhes permite realizar *backups* sem parar a execução do sistema (*hot backup*) e também possibilita a introdução de mais nós de acordo com a necessidade de aumento da capacidade.

Como todas as soluções de bancos de dados disponíveis no mercado, CouchDB possui prós e contras que, dependendo da necessidade da aplicação, podem ou não ser um problema.

Como todo projeto em desenvolvimento, CouchDB ainda precisa de alguns ajustes em suas ferramentas para que se torne mais atrativo e eficiente aos usuários. Operações de *sharding*, indispensáveis no contexto atual de soluções big data, ainda precisam ser melhor especificadas. Ferramentas voltadas para operações de *backup* e *restore* também precisam ser implementadas.

Por garantir consistência eventual, CouchDB não é totalmente recomendado para aplicações intensivas em operações de escrita. No entanto, para aplicações onde operações de escrita não são muito solicitadas o CouchDB atende de forma bastante eficiente.

Aplicações onde a consistência das informações pode ser deixada em segundo plano (redes sociais, blogs etc) são atendidas perfeitamente pelo CouchDB que ainda lhes garante aumento significativo de performance.

Com ferramentas administrativas de fácil utilização, possibilidade de trabalhar com formatos de dados flexíveis, utilitários de configuração e a disponibilização de API REST que garante acesso simples a recursos e dados do banco, CouchDB oferece uma alternativa viável para aplicações que exigem maior distribuição dos dados e maior escalabilidade do banco, tornando-se cada vez mais atrativo para implementação de aplicações voltadas tanto para web como para soluções internas.

# Referências Bibliográficas

ANDERSON, J. C.; SLATER, N.; LEHNARDT, J. **CouchDB: The Definitive Guide**, 1ª edição, O'Reilly Media, 2010.

BREWER, E. A. **Towards Robust Distributed Systems**. Principles of Distributed Computing (PODC), Portland, Oregon, 2000.

BRITO, R. W. **Bancos de Dados NoSQL x SGBDs Relacionais:Análise Comparativa**. Disponível em: [http://www.infobrasil.inf.br/userfiles/27-05-S4-1-68840-Bancos%20de%20Dados%20NoS QL.pdf](http://www.infobrasil.inf.br/userfiles/27-05-S4-1-68840-Bancos%20de%20Dados%20NoS%20QL.pdf). Faculdade Farias Brito e Universidade de Fortaleza, 2010.

BROWN, MC. **Getting Started with CouchDB**, 1ª edição, O'Reilly Media, 2012.

FRIESS, I. I. **Análise de Bancos de Dados NoSQL e Desenvolvimento de uma Aplicação**, Trabalho de Graduação (Curso de Ciência da Computação), Universidade Federal de Santa Maria - Centro de Tecnologia, 2013.

HOLT, B. **Scaling CouchDB**, 1ª edição, O'Reilly Media, 2011.

HOLT, B. **Writing and Querying MapReduce Views in CouchDB**, 1ª edição, O'Reilly Media, 2011.

**The CouchDB Project**. Disponível em: <http://couchdb.apache.org/index.html>. The Apache Software Foundation, 2016.

1. Inicialmente o termo NoSQL sugeria que a solução não utilizava a linguagem SQL. Atualmente essa abordagem não é mais empregada e o termo passou a representar soluções não somente SQL (*Not Only SQL*) [↑](#footnote-ref-1)
2. A VM utilizada neste trabalho encontra-se disponível para download em [https://www.dropbox.com/s/ 44bxejubuh0bumm/CentOS\_7\_CouchDB\_1\_6\_1.ova?dl=0](https://www.dropbox.com/s/%2044bxejubuh0bumm/CentOS_7_CouchDB_1_6_1.ova?dl=0) [**login**: User, **senha**: BlocoCCouchDB, **senha do root**: uni211008] [↑](#footnote-ref-2)
3. CouchDB também permite replicações no padrão *master-master*. Para isso basta configurar duas tarefas de replicação em direções opostas. [↑](#footnote-ref-3)
4. <http://docs.couchdb.org/en/2.0.0/cluster/index.html>. [↑](#footnote-ref-4)