# Um estudo comparativo entre bancos de dados nosql: dynamodb e redis
# A comparative study between nosql: dynamodb and redis databases

[(1)]Jonathan de Carvalho Silva, jonathancs@inatel.br
[(1)]Gimenes Rodrigues Coelho, gimenescoelho@gmail.com
[(1)]Lamon Bese Cambraia, lamoncambraia@gmail.com
[(1)]Vinícius de Oliveira, iniv09@gmail.com

[(1)] Instituto Nacional de Telecomunicações – INATEL, Av. João de Camargo, 510 - Centro, Santa Rita do Sapucaí - MG, 37540-000

Recebido: 28 de fevereiro de 2019; Revisado: 25 de abril de 2019

**Resumo**

A popularização da internet vem resultando em um rápido crescimento do volume de dados, tornando-se cada vez mais complexa a manipulação e manutenção desses dados. Estes dados gerados diariamente implicam a necessidade de adotar um modelo de banco de dados não relacional, NoSQL, esses dados são comumente chamados Big Data. O NoSQL tem o propósito de prover de alta performance e escalabilidade horizontal por meio de um novo conceito de bancos de dados não relacional. Os bancos de dados NoSQL podem ser classificados em orientados a documento, Grafos, Colunas, Chave/Valor. O processamento de uma quantidade tão grande de dados exige um alto processamento de dados do ponto de vista da escalabilidade, distribuição e confiabilidade. Neste trabalho, compara-se dois bancos de dados NoSQL: DyanomoDB e Redis. São bandos do tipo Chave/Valor e suas principais características, de acordo com o teorema CAP, foram apontadas para instruir a melhor escolha, dependendo do ambiente e da demanda de aplicação.

**Palavras-chave**: Banco de dados NoSQL, Chave/Valor, DynamoDB, Redis.

**Abstract**

The popularization of the Internet has resulted in a rapid increase in the volume of data, becoming increasingly complex the manipulation and maintenance of this data. This data generated daily implies the need to adopt a non-relational database model, NoSQL, This data is commonly called Big Data. NoSQL aims to provide high perfomance and horizontal scalability through a new concept of non-relational databases. NoSQL databases can be classified into document-oriented, Graphs, Columns, Key / Value. Processing such a large amount of data requires high data processing from the point of view of scalability, distribution, and reliability. In this work, we compared two NoSQL databases: DyanomoDB and Redis. These are groups of the Key / Value type and their main characteristics, according to the CAP theorem, were pointed out to instruct the best choice depending on the environment and the application demand.

**Key word**: NoSQL database, key-value ,DynamoDB, Redis.

## Introdução

O advento da Web 2.0 marcou o início de uma nova era da produção de dados. Atualmente, a quantidade diária de dados gerados em aplicações Web, redes sociais, redes de sensores entre outras, estão na ordem dos peta bytes (SRIVASTAVA *et al.*, 2015; CARDOSO *et al*., 2012).

Realizar a manipulação, o armazenamento e o processamento dessa massa enorme de dados utilizando os bancos de dados relacionais se mostrou ineficiente, pois o modelo relacional não foi desenvolvido para operar com essa quantidade enorme de informação em um curto intervalo de tempo. Além disso, as aplicações atuais exigem que o sistema seja altamente escalável, possua baixa latência na realização de consultas, suporte modelos flexíveis de armazenamento de dados e simplicidade na distribuição e replicação dos dados (VIEIRA *et al*., 2012). Perante este cenário surgiu o modelo denominado de NoSQL (Not only SQL) que foi proposto para resolver os requisitos do gerenciamento de grandes volumes de dados, semiestruturados ou não estruturados, especialmente na melhoria dos critérios como escalabilidade, controle de concorrência, disponibilidade e custo operacional (VIEIRA *et al*., 2012). Os bancos de dados NoSQL podem ser definidos como um grupo de bancos de dados distribuídos que não obedecem aos princípios dos sistemas relacionais convencionais (VIEIRA et al., 2012). Algumas das suas principais características são definidas a seguir:

**Escalabilidade horizontal:** Com o aumento no volume dos dados surge a necessidade de escalar e melhorar o desempenho do sistema. Dentre as soluções para essa demanda, tem-se a escalabilidade horizontal, que é o aumento do número de máquinas disponíveis para o armazenamento e processamento dos dados.

**Ausência de esquema ou esquema flexível:** a ausência completa ou parcial de esquema que define a estrutura dos dados é uma característica dos bancos NoSQL. Essa falta de esquema simplifica escalabilidade do sistema bem como aumenta a sua disponibilidade.

**Suporte nativo à replicação:** Outra maneira de prover escalabilidade é realizar a replicação nativa dos dados, pois isso ajuda a diminuir o tempo gasto na recuperação das informações. Existem basicamente duas abordagens para realizar a replicação: *Master Slave* e *Multi-Master* (VIEIRA *et al*., 2012).

**API (Application Programming Interface) simples para acesso aos dados**: No NoSQL o foco não está na

forma em que os dados são acessados, mas sim em como são recuperados. O uso de APIs se torna essencial para facilitar o acesso a essas informações, permitindo que qualquer aplicação possa fazer uso do banco de forma rápida e eficiente.

**Consistência eventual:** Essa característica está relacionada ao fato da consistência nem sempre ser mantida entre os vários pontos de distribuição de dados. Ela tem como princípio o teorema de CAP (VIEIRA *et al*., 2012) (*Consistency*, *Availability*, *Partition Tolerance*) que diz que só é possível garantir duas das três propriedades entre consistência, disponibilidade e tolerância à partição (BERNADETTE *et al*., 2011).

Os bancos de dados NoSQL estão divididos em quatro grupos: chave-valor, grafos, família de colunas e documentos. Este artigo tem como objetivo comparar dois bancos chave-valor. O primeiro é DynamoDB, um banco criado pela Amazon para atender a sua demanda e-commerce, e que, devido a sua alta disponibilidade e escalabilidade vem ganhando cada vez mais espaço no mercado. O segundo banco apresentado é o Redis, um sistema de armazenamento em memória que vem sendo adotado por grandes empresas do mercado como Instagram, Twitter e GitHub pelo seu excelente desempenho, vasta documentação e confiabilidade (SRIVASTAVA *et al*., 2015) (BERNADETTE *et al*., 2011). O artigo foi organizado da seguinte maneira: Seção de II à IV abordará os conceitos, suas principais características, instalação e uso de ambos os bancos. A seção V será realizada a comparação dos bancos baseados no teorema de CAP e a seção VI com a conclusão do trabalho.

## Material e Métodos

O procedimento utilizado de metodologia foi baseado em uma pesquisa teórica qualitativa das características dos diferentes bancos de dados NoSQL. As características utilizadas na comparação foram: Modelo Primário, Baseado em nuvem, Estrutura e outros.

## Dynamodb

O banco de dados DynamoDB foi desenvolvido pela Amazon na intenção de atender a demanda de sua plataforma de "e-commerce" tais como: alta disponibilidade, extrema escalabilidade, necessidades de durabilidade, gerenciável em sua totalidade e com serviço baseado na nuvem.

Para entender o DynamoDB é fundamental compreender a estrutura do seu modelo de dados: Tabelas, Itens e

Atributos. As tabelas não precisam ter um esquema fixo como: número de colunas, nome das colunas, tipo de dados, ordem das colunas e tamanho das colunas. É necessário apenas da chave primária fixa, seu tipo de dados e um índice secundário. Caso necessário, os atributos restantes podem ser decididos no tempo de execução. Itens no DynamoDB são registrados individualmente em sua tabela. Pode-se ter qualquer número de atributos em um item. O DynamoDB armazena os itens atribuídos como pares de chave-valor (DESHPANDE *et al*., 2014). O tamanho do item é calculado adicionando o comprimento dos nomes de atributos e seus valores. Principais características são definidas a seguir:

**Totalmente gerenciável**

O DynamoDB lida com todas as necessidades de dimensionamento, pois particiona seus dados de maneira que os requisitos de desempenho sejam atendidos. Qualquer sistema distribuído que comece a escalar é uma sobrecarga para gerenciar, mas o DynamoDB é um serviço totalmente gerenciado (DESHPANDE et al., 2014).

Disponibilidade e tolerância a erro - depois que os dados são carregados, eles são replicados automaticamente em diferentes zonas de disponibilidade dentro de uma região, reduzindo qualquer risco associado a falhas. Portanto, mesmo que seus dados em um datacenter sejam perdidos, sempre haverá um backup em outro datacenter Por padrão, o DynamoDB replica de forma automática e sincronizada seus dados para três datacenters diferentes (DESHPANDE et al., 2014).

**Escalabilidade**

Os dados são distribuídos em vários servidores em várias zonas de disponibilidade automaticamente, conforme o aumento dos dados. O número de servidores pode ser facilmente de centenas a milhares. O DynamoDB segue a arquitetura "*shared-nothing*" (DESHPANDE et al., 2014).

**Baixa latência**

O DynamoDB atende uma taxa de transferência muito alta, fornecendo latência de um dígito em milissegundos, pois usa SSD (*solid-state drive*) para desempenho consistente e otimizado em uma escala muito alta. O DynamoDB não indexa todos os atributos de uma tabela, economizando custos, pois só precisa indexar a chave primária, o que torna as operações de leitura e gravação super-rápidas. Qualquer aplicativo em execução em uma instância do EC2 (*Elastic Compute Cloud*) mostrará a latência de um dígito em milissegundos para um item de tamanho 1 KB. As latências permanecem constantes

mesmo em escala devido à natureza altamente distribuída e aos algoritmos de roteamento otimizados (DESHPANDE *et al*., 2014).

Flexibilidade – Sendo um banco de dados NoSQL, os usuários não são forçados a definirem o esquema da tabela com antecedência. Sendo um armazenamento de dados de valor-chave, permitindo que os usuários decidam quais atributos precisam estar presentes em um item na hora. Cada item de uma tabela pode ter diferentes números de atributos (DESHPANDE *et al*., 2014).

Indexação – A chave primária de cada item é indexada, que por sua vez permite acessar qualquer elemento de maneira mais rápida e eficiente. Também permite índices secundários globais e locais, possibilitando ao usuário consultar qualquer atributo de chave não primária (DESHPANDE *et al.*, 2014).

O DynamoDB é fornecido como serviço pela Amazon AWS, não é necessário instalar para começar a usar, o uso é feito através de um Web Service. Adicionalmente, você pode rodar uma versão local no seu computador para testar sua aplicação sem a necessidade de acessar o Web Service.

**A. Pré-requisitos**

Para o uso local do DynamoDB é necessário ter previamente instalados os seguintes items:

- Java 1.8.
- AWS CLI.

**B. Instalação local do DynamoDB**

Faça o download do DynamoDB, descompacte em um diretório qualquer e coloque para rodar:

```
$java -Djava.library.path=./DynamoDBLocal_lib \ -jar DynamoDBLocal.jar -sharedDb.
```

**C. Trabalhando com o DynamoDB**

Criando a tabela “Carro”:

```
$ aws dynamodb create-table --table-name Carro \
--attribute-definitions \
AttributeName=Marca,AttributeType=S \
AttributeName=Modelo,AttributeType=S \
--key-schema \
AttributeName=Marca,KeyType=HASH \
AttributeName=Modelo,KeyType=RANGE \
--provisioned-throughput \
ReadCapacityUnits=1,WriteCapacityUnits=1 \
--endpoint-url http://localhost:8000
```

Listando as tabelas existentes:

```
$ aws dynamodb list-tables \
--endpoint-url http://localhost:8000
```

Inserindo um item na tabela “Carro”:

```
$ aws dynamodb put-item \
--table-name Carro \
--item '{"Marca": {"S": "Ford"}, \
--return-consumed-capacity TOTAL \
```

"Modelo": {"S": "Ka"}, "Cor": {"S": "Azul"}}' \ --endpoint-url http://localhost:8000

Realizando uma consulta na tabela “Carro”:

```
$ aws dynamodb query --table-name Carro \
--key-condition-expression "Marca = :name" \
--expression-attribute-values \
'{":name":{"S":"Ford"}}' \
--endpoint-url http://localhost:8000
```

**Redis**

Redis é um banco de dados NoSQL de código aberto desenvolvido por Salvatore Sanfiippo. Possui suporte a cinco estruturas de dados distintos sendo elas *strings*, *hashes*, *lists*, *sets* e *sorted sets* (SAMPAIO *et al.,* 2015). No Redis os dados são carregados na memória primária onde todas a operações são realizadas. Dependendo da finalidade de uso, os dados podem ser periodicamente armazenados de maneira assíncrona na memória secundária. Devido a essa característica, o Redis possui uma excelente performance, tornando-o adequado para a realização de cache, gerenciamento de sessões, e classificações (SAMPAIO *et al.*, 2015).

Desse modo, o nó mestre não precisa esperar que o comando seja processado por todos os nós escravos. Quando uma instância mestra e uma escrava está estabelecida, o nó mestre mantém o escravo atualizado enviando um fluxo de comandos para o escravo, a fim de replicar suas alterações (SAMPAIO *et al*., 2015).

**Disponibilidade**

O Redis utiliza arquitetura de réplica principal em um único nó ou em um cluster. Nessa última adota a topologia mestre-escravo onde todos os dados são encaminhados do nó mestre para os nós escravos. Em caso de falha do nó mestre, um dos nós escravos pode ser promovido a mestre através do Redis Sentinel, um sistema distribuído que monitora as condições do cluster (SAMPAIO *et al*., 2015). Consistência – O Redis é capaz de oferecer apenas consistência eventual, visto que não há como garantir, por meio da replicação assíncrona, que um cliente consiga ler de um determinado nó escravo o último valor escrito.

Caso, durante a leitura de um cliente, o nó escravo estiver em defasagem com o nó mestre, o valor lido pelo cliente estará obsoleto. Assim, esse atraso pode acarretar inconsistência por um breve momento.

**Sharding –** Por meio do *Sharding* de dados é possível obter escalonamento horizontal em um cluster Redis. Essa técnica divide todos os dados em várias instâncias do Redis, de forma que cada instância contenha apenas um subconjunto das chaves (SAMPAIO *et al*., 2015).

**A. Instalação do Redis**

Faça o download, descompacte e compile Redis:

```
$ wget http://download.redis.io/ \
releases/redis-5.0.0.tar.gz
$ tar xzf redis-5.0.0.tar.gz
$ cd redis-5.0.0
$ make
```

**B. Colocando Redis para rodar**

Execute o comando:

```
$ src/redis-server
```

**C. Trabalhando com Redis**

Execute o cliente Redis:

```
$ src/redis-cli
```

Inserindo a chave "filme:nome" com o valor "O Hobbit":

```
> SET filme:nome "O Hobbit"
```

Consultando o valor da chave "filme:nome":

```
> GET filme:nome
```

Inserindo múltiplos valores a uma chave em um único comando:

```
> HMSET carro:1 Marca "Ford" Modelo "Ka" Cor
"Azul"
```

Consultando a Marca do carro 1:

```
> HGET carro:1 Marca
```

Consultando todo o conteúdo da chave "carro:1":

```
> HGETALL carro:1
```

**Comparação**

A escolha do banco de dados a ser usado depende da necessidade de cada aplicação ou negócio. O teorema CAP ficou famoso e foi usado como justificativa das compensações entre Consistência, Disponibilidade e Tolerância à Partição feitas pelos engenheiros ao desenvolverem bancos de dados NoSQL. Em 2012, Eric Brewer, autor do teorema, publicou um novo artigo "*CAP 12 Years Later*" (BREWER et al., 2012), em que ele explica alguns equívocos sobre o entendimento do teorema, como, por exemplo, as compensações normalmente são mais sutis que a eliminação da consistência ou da disponibilidade por inteiro.

A Amazon, por exemplo, decidiu que a alta disponibilidade era o mais importante para o seu negócio online. Então, baseado nessa prioridade, fez uma série de sacrifícios para obter alta disponibilidade.

Comparando DynamoDB e Redis nos termos CAP, tem-se o DynamoDB voltado para alta disponibilidade, limitando consultas complexas, adotando estrutura de dados mais simples e introduziu o conceito de "Consistência Eventual" ao invés suportar consistência global. Enquanto o Redis é voltado para consistência eventual, tolerância à partição e possui estrutura de dados mais simples que o DynamoDB.

**Tabela 1.** Comparação de Propriedades

| Parâmetros | DynamoDB | Redis |
|---|---|---|
| Modelo Primário | Documentos Chave-valor | Chave-valor |
| Baseado em Nuvem | Sim | Não |
| Estrutura | Schema-free | Schema-free |
| Particiona | Sharding | Sharding |
| Consistência | Eventual ou Imediata | Eventual ou Imediata |
| Suporte Transacional | Não | Optimistic locking |
| In-memory | Não | Sim |

## Conclusões

Utilizando duas combinações do teorema de CAP é possível definir uma abordagem comparativa para estabelecer qual estrutura e técnica se aplica melhor no contexto da aplicação do banco de dados.

Entretanto, na arquitetura do DynamoDB e do Redis, é preciso ponderar os seguintes pontos para se ter uma melhor resposta em alta demanda: latência, disponibilidade, tolerância a erro e plataforma de serviço.

Esses pontos são decisivos na definição do modelo de banco de dados a ser adotado na aplicação e seus respectivos desafios.

## Referências


SRIVASTAVA, P. P; GOYAL, S.; KUMAR, A. Analysis of various NoSQL database. In 2015 **International Conference on Green Computing and Internet of Things (ICGCIoT).** Noida, Índia, 2015.

CARDOSO, R. M. F, Bases de Dados NoSQL. Mestre em Engenharia Informática, Área de Especialização em Arquiteturas Sistemas e Redes. Instituto Superior de Engenharia do Porto, Porto, 2012.

VIEIRA, M. R., Bancos de Dados NoSQL: conceitos, ferramentas, linguagens e estudos de casos no contexto de Big Data, **Simpósio Brasileiro de Bancos de Dados - SBBD** 2012, São Paulo 2012.

BERNADETTE, L. F.; HÉLIO, O. R., **NoSQL no desenvolvimento de aplicações Web colaborativas**, Juiz de Fora, 2011.

DESHPANDE, T. - **Mastering DynamoDB**, August 2014.

SAMPAIO, P. J. **Desempenho de Aplicações Web: Um estudo comparativo utilizando o software** Redis, Juiz de Fora, 2011.

BREWER, E. CAP **Twelve Years Later: How the "Rules"** Have Changed, 2012.