**Introdução**

Para quem não conhece o Redis, ele é um banco de dados NOSQL (*Not Only SQL*) que trabalha com o conceito de chave / valor. Graças ao fato de armazenar seus dados em memória, ele é considerado ser extremamente rápido.

Para que você possa ter um melhor entendimento sobre esse assunto, eu irei criar alguns exemplos utilizando o Redis com o Node.js.

Para não precisar instalar o redis no meu computador, eu irei utilizar ele dentro de um contêiner Docker.

**Configuração do ambiente**

O primeiro passo será baixar a imagem do redis. Para isso, execute o comando abaixo no seu terminal:

* docker pull redis

O redis utiliza a porta 6379 como default, você pode alterar ela no momento da criação do seu contêiner, mas como esse é um exemplo simples, eu irei utilizar essa porta mapeada com o meu host. Para isso, execute o comando abaixo no seu terminal:

* docker run --name redis13 -p 6379:6379 -d redis redis-server --appendonly no

O comando acima irá criar um contêiner chamado redis13. Para os nossos próximos passos eu irei criar um novo projeto Node.js.

**Criação do projeto**

Escolha um diretório no seu computador, em seguida execute o comando abaixo no seu terminal dentro desse **diretório**:

* npm init -y

O comando acima irá criar um arquivo chamado *package.json*. O próximo passo será baixar o pacote do redis.

*Obs.: Esse comando deve ser executado dentro do diretório que você executou o comando anterior “npm init -y”.*

* npm install redis --save

Com o projeto criado e o pacote do redis importado no projeto, crie um novo arquivo chamado *app.js,* em seguida atualize ele com o seguinte trecho de código:

* var redis = require('redis');
* var client = redis.createClient();
* client.on('connect', function () {
* console.log('connected');
* });

Agora execute o comando ***node app.js***, caso tudo estiver OK, você irá receber uma mensagem no seu console dizendo que você está conectado.

**Armazenando valores**

Agora que você já está conectado com o redis através do seu app node, veja abaixo alguns tipos de valores que você pode armazenar nele.

#### String

O Comando para enviar um valor para o redis é o set. Abaixo você tem um exemplo de como utilizar ele para armazenar uma string.

* client.set('total', '1800');

Agora vamos acessar o cliente do redis dento do contêiner para verificar se o nosso valor foi armazenado corretamente. Para isso, execute o comando abaixo no seu terminal:

* docker exec -it redis13 redis-cli

Agora para retornar o valor de uma chave armazenada no redis, vamos utilizar o get + a [key] que nós criamos.

Tendo o exemplo anterior em mente, digite ***get total*** *no cliente do redis*. Abaixo você tem uma imagem demonstrando esse passo:



#### Hash

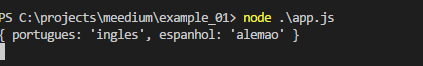
O hash permite você adicionar mais de um valor dentro de uma chave, no exemplo abaixo, eu criei uma chave com o nome languages e adicionei alguns idiomas nela.

* client.hmset('languages', 'portugues', 'ingles', 'espanhol', 'alemao');

Agora para listar todos os registros inseridos nessa chave, vamos utilizar *hgetall:*

* client.hgetall('languages', function (err, object) {
* console.log(object);
* });

Executando o código novamente você irá receber o resultado da imagem abaixo:



#### Lists

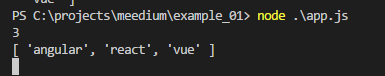
O Redis também nos permite armazenar uma lista de dados. Para ficar mais claro, vamos a mais um abaixo:

* client.rpush(['frameworks', 'angular', 'react', 'vue'], function (err, reply) {
* console.log(reply);
* });

O exemplo acima irá criar uma lista chamada frameworks com 3 registros. Agora para retornar esses valores, nós podemos utilizar o lrange. Veja abaixo um exemplo demonstrando esse passo:

* client.lrange('frameworks', 0, -1, function (err, reply) {console.log(reply);
* });

Resultado:



#### Sets

Ainda falando sobre listas, nós podemos utilizar o set para inserir uma lista no nosso banco de dados, a diferença do set para list do exemplo anterior, é que ele não permite duplicar um registro. Veja abaixo um exemplo:

* client.sadd(['tags', 'angularjs', 'angularjs', 'backbonejs', 'emberjs'], function (err, reply) {
* console.log(reply);
* });

Agora para resgatar os dados do set e validar se ele inseriu um registro duplicado, você pode utilizar o ***smembers.*** Veja abaixo um exemplo demonstrando esse passo:

* client.smembers('tags', function (err, reply) {
* console.log(reply);
* });

Note que ele retornou a nossa lista sem duplicar o valor "angularjs"

Resultado:



#### Del e Expire

Em alguns momentos você vai precisar deletar ou expirar o valor de uma chave. Para isso, o redis permite utilizar o **del** e o **expire**. Veja abaixo dois exemplos demonstrando a utilização deles.

**Expire**

Para demonstrar o expire eu irei utilizar o primeiro exemplo do artigo com uma string.

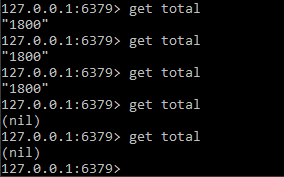
Abaixo você tem um exemplo demonstrando como expirar a nossa chave *total* depois de 10 segundos:

* client.set('total', '1800');
* client.expire('total', 10);

Para validar esse passo, execute novamente *node app.js*, em seguida conecte novamente no seu redis-cli e execute o comando abaixo:

* 127.0.0.1:6379> get total

Note que depois de 10 segundos você irá receber o resultado abaixo:



**Del**

Por fim o del remove o registro no momento que você executa ele. Veja abaixo um exemplo utilizando a chave frameworks.

* client.rpush(['frameworks', 'angular', 'react', 'vue'], function (err, reply) {
* console.log(reply);
* });
* client.del('frameworks', function(err, reply) {
* console.log(reply);
* });
* client.smembers('tags', function (err, reply) {
* console.log(reply);
* });

Abaixo você tem uma imagem demonstrando o resultado desse passo:

