MIT em Big Data

Bloco C – Armazenamento Heterogêneo de Dados

Trabalho de Bloco

Banco de Dados chave-valor em memória

Componentes do Grupo

Carlos Oliveira

Carlos Toledo

Debora Serôdio

Nathalia Dias

Sumário

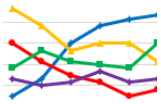
O que é o Redis?

Redis é um datastore NoSQL de código aberto (licenciado pelo BSD) do tipo chave-valor. Conforme ranking do site DB-Engines, Redis é o 7° banco mais utilizado e o primeiro no ranking de modelo chave-valor.

DB-Engines Ranking

The DB-Engines Ranking ranks database management systems according to their popularity. The ranking is updated monthly.

Read more about the method of calculating the scores.



trend chart

342 systems in ranking, May 2018

Rank May 2018	Rank Apr 2018	Rank May 2017	DBMS	Database Model	Score May 2018	Score Apr 2018	Score May 2017
1.	1.	1.	Oracle	Relational DBMS	1290.42	+0.63	-63.90
2.	2.	2.	MySQL	Relational DBMS	1223.34	-3.06	-116.69
3.	3.	3.	Microsoft SQL Server	Relational DBMS	1085.84	-9.67	-127.96
4.	4.	4.	PostgreSQL	Relational DBMS	400.90	+5.43	+34.99
5.	5.	5.	MongoDB	Document store	342.11	+0.70	+10.53
6.	6.	6.	DB2	Relational DBMS	185.61	-3.34	-3.23
7.	↑ 9.	↑ 9.	Redis	Key-value store	135.35	+5.24	+17.90

***Img_01:** DB-Engines Ranking*

O modelo chave-valor é um sistema onde os dados são armazenados em forma de pares chave e valor, como por exemplo:

nome= ”Augusto”

profissao= [“analista”, “desenvolvedor”]

No exemplo acima, nome e profissão são as chaves e seus valores estão sendo declarados logo à direita.

O armazenamento dos dados no Redis é em memória (in-memory), ao contrário de bancos de dados conhecidos como: PostgreSQL, MongoDB e outros que armazenam os dados, na sua grande maioria, em disco.

Para atingir seu máximo em performance, o Redis trabalha com datasets armazenados em memória garantindo assim suporte a um número imensamente maior de operações e tempo de resposta mais rápidos. Todas as operações no Redis são atômicas isso significa que enquanto uma operação for executada, nenhuma outra é executada paralelamente.

Um ponto interessante do Redis, é que apesar dos dados serem armazenados em memória ele persiste os dados em disco devido seu tratamento aplicado à failback. Ele consegue fazer dois tipos de persistência:

- dump binário: De tempos em tempos, os dados são persistidos em disco. O período em que o dump será executado, pode ser programado.
- log de operações: Todos os comandos executados são salvos e quando necessário, os mesmos são re-executados.

Por padrão, o método de dump binário está ativo. Ele está salvo em um arquivo chamado dump.rdb.

Aplicações

Devido ao alto desempenho e facilidade de uso do Redis, ele acaba se tornando uma escolha em alta demanda para aplicações web e móveis, como também a IoT que exigem o melhor desempenho do mercado.

O Redis pode ser utilizado como:

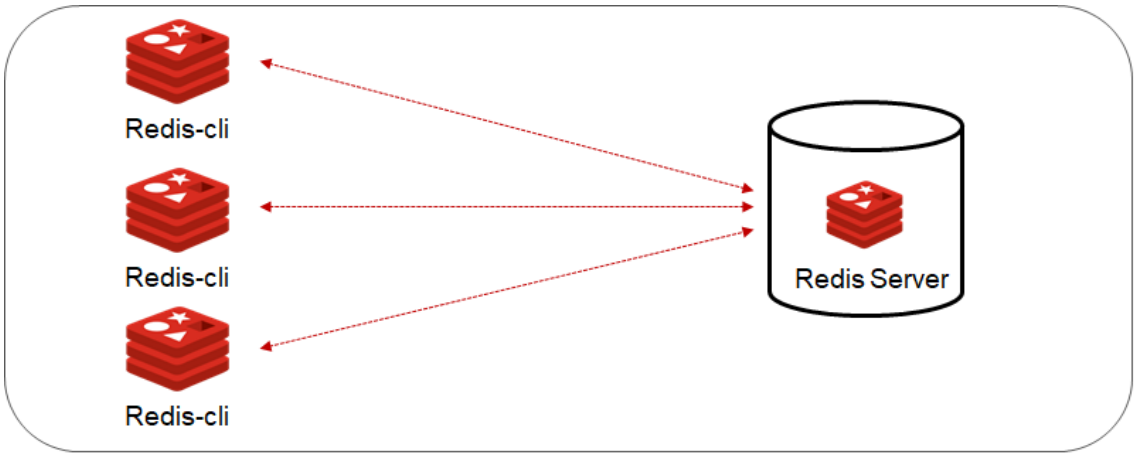
- **base de dados**: com base em bibliotecas de ODM (Object Data Mapper), utiliza as estruturas internas para poder criar tabelas, índices e realizar consultas;
- **cache de dados**: os dados são armazenados e recuperados de forma mais rápida. Serve tanto para web quanto para aplicações;
- **tabela de classificação**: o tipo de dados “sorted set” permite armazenar pares de valor e scores que são utilizados para implementação de jogos e listas de classificação. Permite atualizar uma tabela ordenando as chaves de acordo com o score armazenado.
- **gerenciamento de sessão**: o id e todas as informações de sessão são armazenados no tipo de dados “hash”, dessa forma o Redis permite que 1 ou múltiplos campos sejam retornados e/ou modificados. Utilizando em par com a função “expire”, a sessão pode ser expirada por completo apagando o “hash” que a representa uma única vez;

Oferece suporte às linguagens: Python, Java, Ruby, Lua, JavaScript dentre outras.

Arquitetura

Contém dois processos principais: Redis Client e Redis Server. Tanto o Redis Client quando o Redis Server, podem estar no mesmo computador ou em dois computadores diferentes.

Redis Server é responsável por armazenar os dados em memória. Responsável por cuidar de todos os tipos de gerenciamento e compor a maior parte da arquitetura, já o Redis Client pode ser tanto o cliente do console ou a API Redis de qualquer outra linguagem de programação.



Img_02: *Redis cliente - servidor*

Suporte a transações ACID

Redis, não sendo o banco de dados relacional, não é suposto a suportar transações ACID. Não há mecanismo de Rollback no Redis, no entanto em relação às propriedades do ACID:

- Atomicidade e consistência podem ser garantidas para um grupo de comandos server-side com scripts Lua;
- Isolação é sempre garantida a nível de comando, e pode ser garantida para um grupo de comandos usando um bloco MULTI / EXEC ou Lua;
- Durabilidade pode ser garantida quando AOF está ativado (com fsync sistemático).

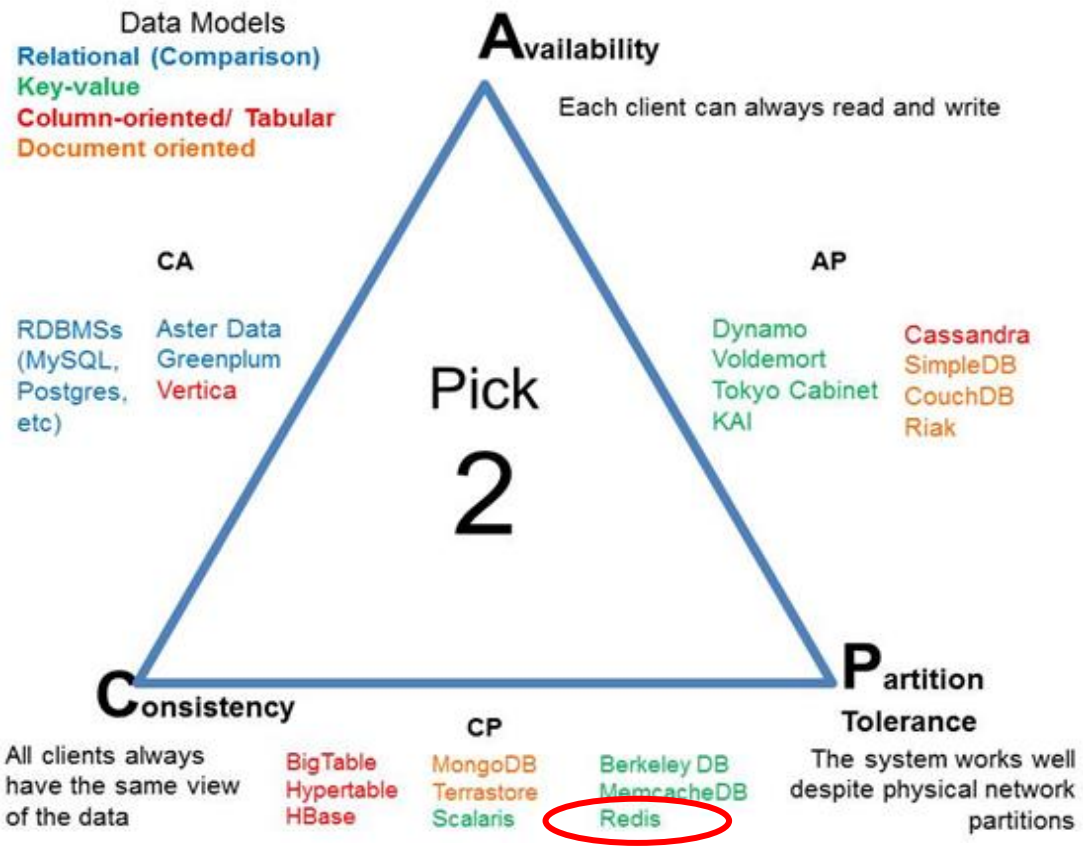
Teorema de CAP

O Teorema do CAP, também chamado de **Teorema de Brewer**, afirma que é impossível que o armazenamento de dados distribuído forneça simultaneamente mais de duas das três garantias seguintes:

- Consistência: Cada leitura recebe a escrita mais recente ou um erro
- Disponibilidade (Availability) cada pedido recebe uma resposta (sem erro) - sem garantia de que contém a escrita mais recente
- Particionamento O sistema continua a funcionar apesar de um número arbitrário de mensagens serem descartadas (ou atrasadas) pela rede entre nós

Em outras palavras, o teorema de CAP afirma que, na presença de uma partição da rede, é preciso escolher entre consistência e disponibilidade.

Em relação ao teorema de CAP o Redis é classificado como CP, ou seja, no caso de partição da rede e escolha do Redis é em relação a Consistência, sacrificando a disponibilidade.



Na realidade quando em cluster o Redis em um primeiro momento escolhe a disponibilidade (Avaiability) quando alguns nodes falham ou para se comunicar, no entanto nos casos onde a maioria dos masters não está disponível o cluster para de operar, passando para uma condição de CP em relação ao teorema de CAP.

Não é possível para o Redis garantir uma consistência forte, o que significa que em certas condições é possível que o cluster Redis perca algumas escritas, como por exemplo quando um node recebe escritas e cai antes que possa sincronizar com sua réplica, neste caso a réplica assume o lugar do master sem o dado escrito no master. Um dos modos de garantir a consistência é fazer com que o Redis sempre grave em disco (flush) antes de cada operação de replicação, no entanto este procedimento traria grandes impactos na performance.

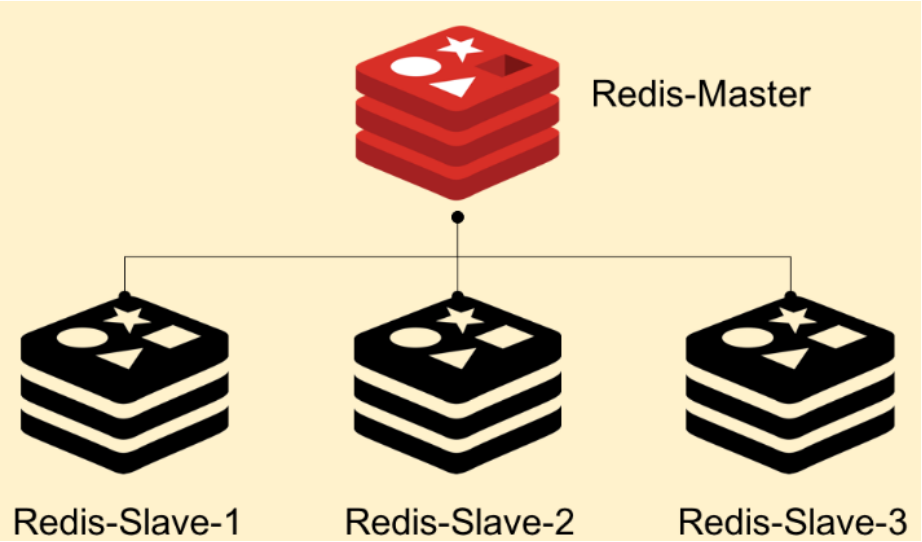
Clusterização

Redis permite realizar cluster de forma Master-Slave, o que permite atingir os seguintes objetivos:

- Desabilitar a persistência no Master node, permitindo um tempo de resposta consistente de baixa latência. Sem perda de I/O;
- Entrega de alta disponibilidade: se o Master node ficar offline, o slave node pode ser automaticamente promovido a master node de forma que não ocorra tempo offline;
- Alto throughput: Sob extrema carga, as leituras podem ser equilibradas entre o master node e slave node.

Quando o cluster é criado, o Redis não suporta múltiplos databases como na versão stand alone. Apenas o database 0 é habilitado e o comando “select” não é permitido.

Para manter a disponibilidade do sistema, cada node master terá de um a N réplicas.



Img_03: *Redis Master - slave*

O cluster do Redis provê um modo no qual o dado é automaticamente "shardeado" através dos múltiplos nodes do cluster. Cada chave no Rest cluster é parte de um Hash Slot, o total de Hash Slots em um cluster Redis é de 16.384 hash slots. Este número total de Hash Slots é distribuído pelo nós master do cluster.

Para auxiliar na obtenção de alta disponibilidade, Redis disponibiliza o Sentinel. De forma prática, o Sentinel permite implantar o Redis de forma que ele resista sem a necessidade de intervenção humana para atuação em certos tipos de falha.

Instalação

O *Redis* possui três canais principais de distribuição.

1. Distribuição estável. É o último release estável distribuído.
2. Distribuição instável. São versões em desenvolvimento com novas features ou melhorias em desempenho que podem ser importantes, porém há um risco em se utilizar estas versões, por não terem sido completamente testadas e aprovadas.
3. Docker. Os releases mais recentes são imediatamente disponibilizados em formato docker.

A distribuição do Redis é produzida apenas para a plataforma linux. Para produção é recomendado apenas a plataforma de 64 bits do Ubuntu, Red Hat, Oracle e Amazon Linux, a versão Enterprise tem limitações adicionais para quais versões de linux é suportada. Windows não é uma plataforma suportada, e apesar de existirem roteiros

para instalação no windows, eles não são recomendados nem suportados pela comunidade Redis e por experiência sujeitos a falhas.

A instalação é extremamente simples. Realiza-se o download e descompressão da versão desejada e efetua-se a compilação do código fonte. Os binários estão gerados.

$ wget http://download.redis.io/releases/redis-4.0.9.tar.gz
$ tar xzf redis-4.0.9.tar.gz
$ cd redis-4.0.9
$ make

Neste momento o servidor já pode ser iniciado a partir do seu binário.

$ src/redis-server.

Com o servidor iniciado também se pode interagir com o mesmo utilizando o cliente gerado pela compilação dos “fontes”.

$ src/redis-cli
redis> set foo bar
OK
redis> get foo "bar"

Por padrão, Redis utiliza um arquivo para dump dos dados atualizados, e se conecta a todas as interfaces de rede e não utiliza nenhum tipo de autenticação. A iniciação e finalização do serviço de banco são manuais.

Em ambiente produtivo, deve-se:

1. Considerar o uso de um servidor seguro e isolado da internet por firewall.
2. Realizar as configurações de start e shutdown do banco junto ao servidor.
3. Configurar a porta de acesso ao banco.
4. Configurar quais interfaces de rede serão ouvidas pelo banco.
5. Configurar senha de acesso ao banco.
6. Restringir o acesso ao arquivo de configuração e suas cópias.
7. Utilizar ferramentas de SSL ou túnel para criptografar as comunicações de/para o banco, se for requerimento de negócio

Após a geração de binários, passos adicionais que permitem configurar algumas das recomendações acima seguem abaixo:

1 - Criar diretórios para armazenar os dados do Redis (dump) e os logs.

sudo mkdir /etc/redis
sudo mkdir /var/redis

2 - Copiar o script de inicialização sob utils para /etc/init.d/ sugere-se usar a porta no nome do script.

sudo cp utils/redis_init_script /etc/init.d/redis_6379

3 - Editar o script e ajustar o número da porta, o nome do arquivo de pid e do arquivo de configuração dependente do número da porta.

sudo vi /etc/init.d/redis_6379

4 - Copiar o arquivo de configuração na raiz do diretório do redis recém compilado para o /etc/redis com o número da porta no nome do arquivo.

sudo cp redis.conf /etc/redis/6379.conf

5 - Criar o diretório em /var/redis que servirá como diretório de dados e de trabalho para a instância do Redis.

sudo mkdir /var/redis/6379

6 - Editar o arquivo de configuração onde deverá realizar as seguintes mudanças:

a- Alterar daemonize para sim.

b - Alterar o pidfile para /var/run/redis_6379 (ajustar porta)

c - Manter coerentes todas as alterações que incluem a porta, pois neste exemplo é usada a padrão: 6379

d - Ajustar o nível desejado de loglevel.

e - Alterar o logfile para /var/log/redis_6379.log

f - Alterar dir para /var/redis/6379

g - Adicionar o novo script de init do Redis para todos os runlevels padrão usando o seguinte comando:

sudo update-rc.d redis_6379 defaults

Finalmente pode inicializar a sua instância com:

sudo /etc/init.d/redis_6379 start

Interação

O Redis é banco de dados NoSQL orientado a chave valor. Seus dados são armazenados em memória sendo replicado em disco com isso o redis possui uma excelente performance em comparação a banco de dados relacionais.

Todas as operações no Redis são atômicas isso significa que enquanto uma operação for executada, nenhuma outra é executada paralelamente.

Comandos Básicos

Set

O comando SET é similar ao insert nos bancos de dados relacionais.

```
> SET server:name "fido”
OK
```

inserimos no Redis uma chave chamada “server:name” com valor fido e teremos o retorno “OK”.

Get

O comando GET é similar ao select nos bancos de dados relacionais.

```
> GET server:name
"fido"
```

Retornamos o valor da chave “server:name”. Caso a chave não exista teremos como retorno o valor null ex:

```
> GET teste
(nil)
```

Del

Com o comando DEL iremos excluir uma determinada chave e seu valor associado.

```
> SET chave_exemplo deletar
OK
> GET chave_exemplo
"deletar"
> DEL chave_exemplo
(integer) 1
> GET chave_exempo
(nil)
```

INCR

O comando INCR incrementa atomicamente um número armazenado em uma determinada chave.

```
> SET chave_exemplo 10
OK
> GET chave_exemplo
"10"
> INCR chave_exemplo
(integer) 11
> INCR chave_exemplo
(integer) 12
> INCR chave_exemplo
(integer) 13
```

SETNX

Retorna se a chave existe 1 ou não 0.

```
> SETNX mykey Hello
(integer) 1
> SETNX mykey Hello
(integer) 0
```

Expiração de uma Chave

No redis podemos limitar o tempo de vida de uma chave, isso é feito com os seguintes comandos EXPIRE e TTL.

No comando expire é possível limitar o tempo de vida da chave em segundos. No exemplo abaixo criamos a chave “resource:lock” com valor “Redis Demo” com tempo de expiração em 120 segundos.

```
> SET resource:lock 'Redis Demo'
OK
> EXPIRE resource:lock 120
(integer) 1
```

O comando TTL retorna em segundos o tempo restante vida da chave

```
> TTL resource:lock
(integer) 111
> TTL resource:lock
(integer) 103
> TTL resource:lock
(integer) 92
> TTL resource:lock
(integer) 19
```

Quando o comando TTL retornar -2 significa que a chave não existe mais.

```
> TTL resource:lock
(integer) -2
```

Quando o comando retorna -1 significa que esta chave não possui tempo de vida definido.

```
> TTL chave_exemplo
(integer) -1
```

Tipos de Dados

Strings

O tipo de dados mais básico, Redis é capaz de guardar qualquer valor em formato de string em uma chave.

Listas em Redis

Listas são uma série de valores ordenados. Alguns dos comandos importantes para interagir com listas são: RPUSH, LPUSH, LLEN, LRANGE, LPOP e RPOP.

RPUSH

Insere valores ao final da lista. Se a chave não existir, ela será criada.

```
> RPUSH mylist hello
(integer) 1
> RPUSH mylist world
(integer) 2
```

LRANGE

Retorna os elementos especificados de uma lista a partir do range início e fim. Os valores negativos representam o fim da busca.

```
> LRANGE mylist 0 0
1) "hello"
```

LPUSH

Insere valores ao início da lista. Se a chave não existir, ela será criada.

```
> LPUSH mylist first
(integer) 3
```

Lista todos os elementos da lista

```
> LRANGE mylist 0 -1
1) "first"
2) "hello"
3) "world"
```

LLEN

Retorna o tamanho de uma lista.

```
> LLEN mylist
(integer) 3
```

LPOP

Remove e retorna o primeiro elemento da lista

```
> LRANGE mylist 0 -1

1) "first"
2) "hello"
3) "world"

> LPOP mylist

"first"

> LRANGE mylist 0 -1

1) "hello"
2) "world"
```

RPOP

Remove e retorna o último elemento da lista

```
> RPOP mylist

"world"

> LRANGE mylist 0 -1

1) "hello"
```

Sets

Sets são estruturas similares a lista, porém sets não possuem ordenação e cada elemento só pode aparecer uma única vez. Alguns dos comandos importantes para trabalharmos com Sets são: SADD, SREM, SISMEMBER, SMEMBERS e SUNION.

SADD

Adiciona valores na estrutura Set.

```
> SADD superpowers flight

(integer) 1

> SADD superpowers 'x-ray vision'

(integer) 1

> SADD superpowers reflexes

(integer) 1
```

SREM

Remove um elemento específico da estrutura Set.

```
> SREM superpowers reflexes

1
```

SMEMBERS

Retorna todos os elementos de um Set**.**

```
> SMEMBERS superpowers

1) "flight"
2) "x-ray vision"
```

SISMEMBER

Esse comando testa se o valor existe na estrutura Set. Se o valor existir retorna 1 caso contrário retorna 0.

```
> SISMEMBER superpowers flight

(integer) 1

> SISMEMBER superpowers reflexes

(integer) 0
```

UNION

Combina 2 ou mais sets e retorna todos os elementos.

```
> SADD birdpowers pecking

(integer) 1

> SADD birdpowers flight

(integer) 1

> SUNION superpowers birdpowers

1) "pecking"
2) "flight"
3) "x-ray vision"
```

Sorted Sets

Os Sets são tipo de dados muito práticos, porém não possuem ordenação. Na versão Redis 1.2 foi introduzido os Sorted Sets que nada mais é que o set ordenado onde cada valor tem uma pontuação associada. Essa pontuação é usada para classificar os elementos no set.

```
> ZADD hackers 1940 'Alan Kay'

(integer) 1

> ZADD hackers 1906 'Grace Hopper'

(integer) 1

> ZADD hackers 1953 'Richard Stallman'

(integer) 1

> ZADD hackers 1965 'Yukihiro Matsumoto'

(integer) 1
```

Nestes exemplos, as pontuações são anos de nascimento e os valores são os nomes de hackers famosos.

ZRANGE

Retorna os elementos especificados de um Set a partir do range início e fim.

```
> ZRANGE hackers 2 4

1) "Richard Stallman"
2) "Yukihiro Matsumoto"
```

Os elementos são listados de acordo com a ordenação da pontuação.

Hashs

São mapas entre campos de string e valores de string, então eles são o tipo de dados perfeito para representar objetos (por exemplo: Um usuário com vários campos como nome, sobrenome, idade e assim por diante)

HSET

Insere valores no objeto hash.

```
> HSET user:1000 name 'John Smith'

(integer) 1

> HSET user:1000 email 'john.smith@example.com'

(integer) 1

> HSET user:1000 password 's3cret'

(integer) 1
```

HGETALL

```
> HGETALL user:1000

1) "name"
2) "John Smith"
3) "email"
4) "john.smith@example.com"
5) "password"
6) "s3cret"
```

HMSET

Para inserirmos vários campos de uma vez só.

```
> HMSET user:1001 name 'Mary Jones' password 'hidden' email 'mjones@example.com'

OK
```

HGET

Para retornarmos apenas um campo do objeto hash.

```
> HGET user:1001 name

"Mary Jones"
```

HINCRBY

O comando HINCRBY incrementa atomicamente um número armazenado em uma determinada chave no objeto hash.

```
> HINCRBY user:1000 visits 1

(integer) 1

> HINCRBY user:1000 visits 10

(integer) 11
```

HDEL<key> field [field ...]

Remove os campos especificados do hash armazenado na chave.

```
> HDEL user:1000 password

(integer) 1

> HGETALL user:1000

1) "name"
2) "John Smith"
3) "email"
4) "john.smith@example.com"
5) "visits"
6) "11"
```

HEXISTS <key> field

Retorna se o campo existe no hash armazenado na chave. Se o campo existe retorna 1 se não existir 0.

```
> HSET myhash field1 foo

(integer) 1

> HEXISTS myhash field1

(integer) 1

> HEXISTS myhash field2

(integer) 0
```

SORT

Ordenação de numeros. Retorna ou armazena os elementos contidos na lista, define ou classifica o set na chave. Por padrão, a classificação é numérica e os elementos são comparados por seu valor interpretado como número de ponto flutuante de precisão dupla. Este é o SORT na sua forma mais simples.

```
> RPUSH myListNumbers 1 2 3

(integer) 3

> SORT myListNumbers

1) "1"
2) "2"
3) "3"

> SORT myListNumbers desc

1) "3"
2) "2"
3) "1"
```

SORT alpha

Ordenação de string em ordem crescente.

```
> SADD xurl Facebook.com Buddy.com Yahoo.com

(integer) 3

> SORT xurl alpha

1) "Buddy.com"
2) "Facebook.com"
3) "Yahoo.com"

> SORT alpha desc

(empty list or set)
```

Ordenação com limite

```
> SORT xurl alpha limit 0 2

1) "Buddy.com"
2) "Facebook.com"

> SORT xurl alpha limit 0 10

1) "Buddy.com"
2) "Facebook.com"
3) "Yahoo.com"

> SORT xurl alpha limit 2 10

1) "Yahoo.com"
```

OBJECT ENCODING <key>

Retorna o tipo interno usado para armazenar o valor associado a uma chave.

Objetos podem ser codificados como:

- **Strings**: Podem ser codificadas como **embstr** (codificação normal de uma string) ou **int** (strings que representam inteiros).
- **Lists:** podem ser codificadas como **ziplist** ou **linkedlist**, **quicklist**.
- **Sets** : Podem ser codificados como **intset** ou **hashtable**. O **intset** usado para objetos menores.
- **Hashs:** Podem ser codificadas como **ziplist** ou **hashtable**. O **ziplist** usado para objetos menores.
- **Sorted Sets:** Podem ser codificados como **ziplist** ou **skiplist**. O **Ziplist** usado para objetos menores.

Segurança

Por padrão, o Redis não requer nenhuma autenticação, ele foi projetado para ser acessado por clientes confiáveis em ambientes confiáveis.

O redis fornece uma pequena camada de autenticação que é ativada opcionalmente editando o arquivo redis.conf. Quando a camada de autorização estiver habilitada, o Redis recusará qualquer consulta por clientes não autenticados.

redis.conf

bind 127.0.0.1 // Remove # (comentário) da linha

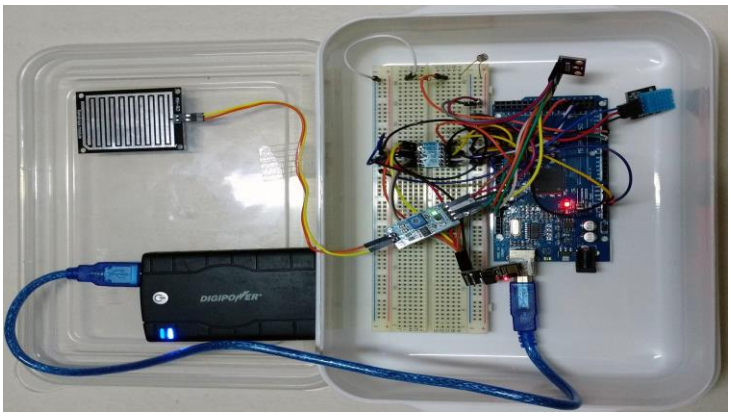
Um cliente pode se autenticar enviando o comando AUTH seguido pela senha. O comando AUTH, como todos os outros comandos do Redis, é enviado sem criptografia, portanto, não protege contra um invasor que tenha acesso suficiente à rede para realizar a interceptação.

127.0.0.1:6379> AUTH password

Evidências

Como teste prático para corroborar o detalhamento teórico, foi realizado a instalação do Redis como banco de dados de dispositivo de IoT (Internet of Things). O experimento

de prova-de-conceito realizou coleta de dados meteorológicos locais com Arduino e armazenamento em Redis para análise de variação, correlação e predição de resultados.



Img_04: *Foto do experimento com Arduino*

Os dados de data hora, temperatura, pressão, altitude, umidade, pressão no nível do mar, índice de luminosidade, índice de calor, ocorrência de chuva, e nível de chuva são coletados pelo dispositivo IoT e persistidos em uma base de dados Redis.

Para inserir dados os dados meteorológicos foram usados a instrução HMSET. Esta instrução, disponível a partir da versão 2.0.0 do Redis, possui a estrutura **HMSET chave campo valor [campo valor ...]** e atribui valores aos campos pertencentes a uma chave. Este comando sobrescreve qualquer campo existente no HASH. Se a chave não existir, uma chave com os pares de campo e valor é criada.

Para armazenar os dados coletados de temperatura, pressão, umidade, índice de luminosidade e nível de chuva foi criado no Redis uma estrutura HASH com o seguinte esquema:

HASH: weather:25311

row	key	value
1	date	05/05/2018 12:58:05 am
2	time	1525492685
3	temperature	26.69
4	pressure	1014.55
5	altitude	45.17
6	sealevelpressure	1020.00
7	humidity	57.00
8	drytemperature	23.00
9	heatindex	22.84
10	rain	1.00
11	rainlevel	883.00
12	lightindex	8.00

A figura anterior apresenta os campos e valores para a chave **weather:25311** do objeto do tipo HASH armazenado no Redis.

Instrução Redis para a inserção de registros no HASH:

```
r->HMSET (“weather:$collid” ,
        ”data” ,  $date ,
        “time” , time() ,
        “temperature” , $temperature ,
        “pressure” , $pressure ,
        “altitude” , $altitute ,
        “sealevelpressure” , $sealevelpressure ,
        “humidity” , $humidity ,
        “drytempeature”, $drytemperature ,
        “heatindex” , $heatindex ,
        “rain” , $rain ,
```

“rainlevel” , $rainlevel ,

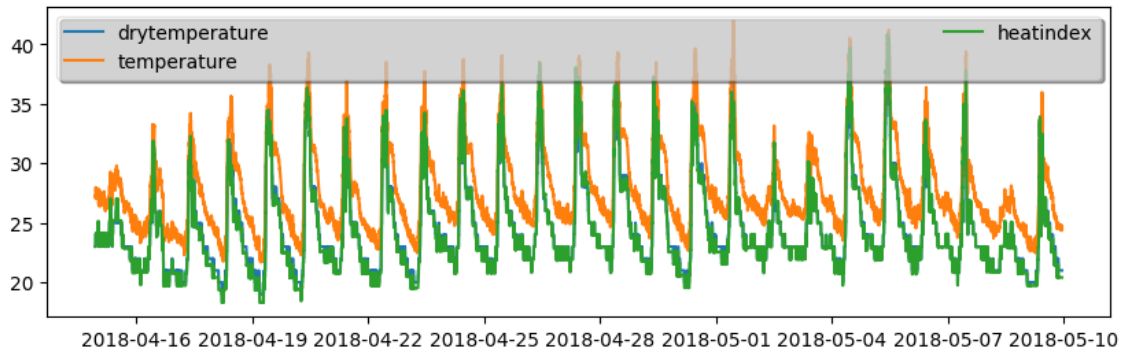
“lightindex” , $lightindex);

A figura abaixo exemplifica uma leitura full dos dados armazenados para gerar modelo preditivo ou gráficos.

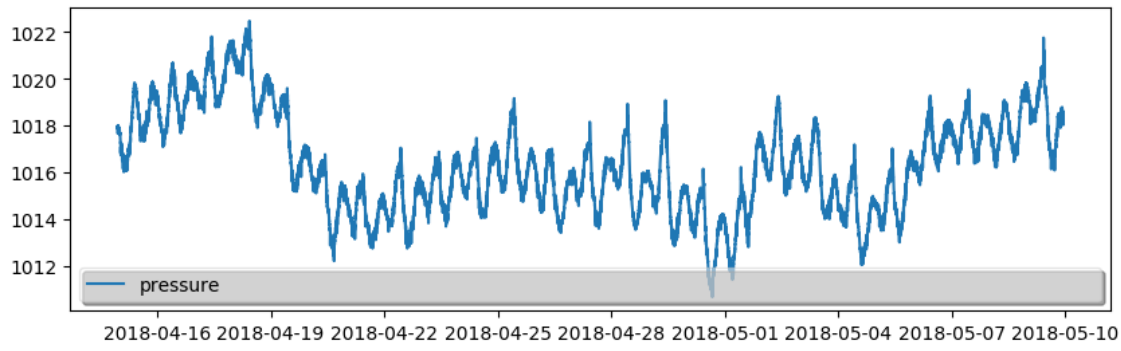
```
# create a connection to the localhost Redis server instance, by
# default it runs on port 6379
redis_db = redis.StrictRedis(host="192.168.1.103", port=6379, db=0)
features = []
records = []
print('Started reading all data from redis')
for key in redis_db.keys('weather:*'):
    features = [item.decode('utf8') for item in list(redis_db.hgetall(key.decode('utf8')).keys())]
    records.append([item.decode('utf8') for item in list(redis_db.hgetall(key.decode('utf8')).values())])
print('Ended reading all data from redis')
df0 = pd.DataFrame(records, columns=features)
```

A partir deste tipo de leitura de dados meteorológicos armazenados no Redis foi possível a geração dos gráficos exemplo abaixo.

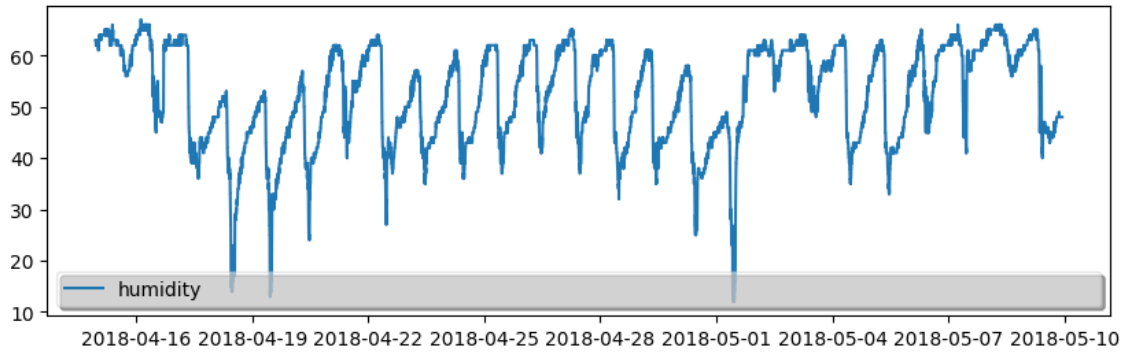
Temperatura



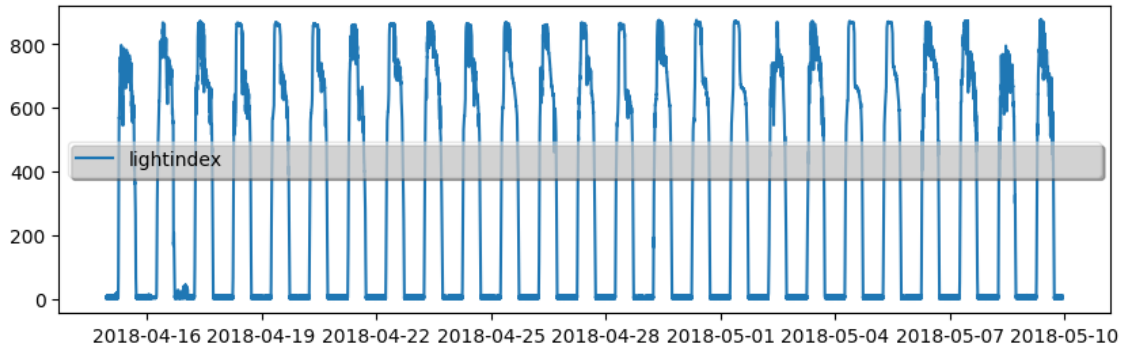
Pressão



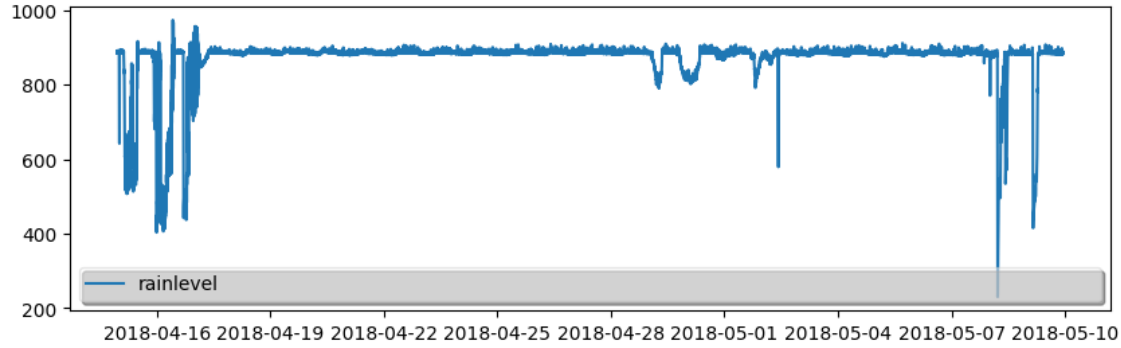
Umidade



Índice de luminosidade



Nível de chuva



Conclusões

Podemos concluir que redis é um banco de dados versátil e robusto pois permite persistência em disco, replicação e compatível com diversas linguagens de programação, porem devido sua simplicidade na segurança sua melhor utilização se por armazenamento em cache, gerenciamento de sessões, chat, sistema de mensagens e classificações em tempo real.

Referências Bibliográficas

Fontes:

https://aws.amazon.com/pt/redis/

https://redis.io/topics/introduction

http://qnimate.com/overview-of-redis-architecture/