

**ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH  
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN  
KHOA HỆ THỐNG THÔNG TIN**



**TIỂU LUẬN  
MÔN CƠ SỞ DỮ LIỆU PHÂN TÁN  
NoSQL và Cơ sở dữ liệu Redis**

Lớp	<b>IS211.M11.HTCL.2</b>	
Giảng viên hướng dẫn	<b>Nguyễn Minh Nhật</b>	
Sinh viên thực hiện	<b>Nguyễn Thị Thúy Nga</b>	<b>MSSV: 19521881</b>
	<b>Võ Trọng Hoàn</b>	<b>MSSV: 19520556</b>
	<b>Lê Quang Huy</b>	<b>MSSV: 19521616</b>

**TP Hồ Chí Minh, tháng 12 năm 2021**

# MỤC LỤC

<b>MỤC LỤC</b>	<b>2</b>
1. Đặt vấn đề	4
2. Cơ sở dữ liệu NoSQL	5
2.1. Tổng quan	5
2.2. Đặc điểm	5
2.3. Tính năng	6
2.3.1. ACID free	6
2.3.2. BASE	6
2.3.3. CAP	7
2.4. Phân loại	8
2.4.1. Key-values Stores	8
2.4.2. Column Family Stores	9
2.4.3. Document Store Databases	10
2.4.4. Graph Databases	12
3. Hệ quản trị cơ sở dữ liệu Redis	15
3.1. Tổng quan về Redis	15
3.2. Đặc trưng của Redis	15
3.2.1. Data Model	15
3.2.2. Master/Slave Replication	17
3.2.3. In-memory	17
3.2.4. Redis Persistence	17
3.3. Cấu trúc dữ liệu của Redis	20
3.3.1. Strings	20
3.3.2. Lists	24
3.3.3. Sets	26
3.3.4. Sorted Sets	28
3.3.5. Hashes	30

3.4. Cơ chế phân tán .....	32
3.4.1. Giao tiếp giữa các node .....	32
3.4.2. Phân vùng dữ liệu .....	33
3.4.3. Mô hình Master-Slave Replication.....	36
3.5. Ưu điểm và nhược điểm .....	37
3.5.1. Ưu điểm .....	37
3.5.2. Nhược điểm .....	38
3.5.3. Ứng dụng .....	38
4. Cài đặt và thực nghiệm mô hình phân tán bằng redis .....	39
4.1. Cài đặt Redis.....	39
4.2. Thực hiện phân tán .....	44
4.3. Thực nghiệm mô hình phân tán.....	48
4.3.1. Mô tả bài toán .....	48
4.3.2. Các bước thực nghiệm.....	49
5. Cơ chế nhân bản .....	52
GITHUB .....	56
PHÂN CÔNG CÔNG VIỆC .....	56
TÀI LIỆU THAM KHẢO .....	57

## 1. ĐẶT VẤN ĐỀ

Hệ quản trị cơ sở dữ liệu quan hệ truyền thống ra đời năm 1970 được thiết kế để phù hợp với các yêu cầu của các ứng dụng xử lý giao dịch trực tuyến (OLTP) như là các giải pháp “một kích thước phù hợp với tất cả”. Các hệ thống này thường được lưu trữ trên một máy chủ duy nhất, nơi quản trị cơ sở dữ liệu đáp ứng với sự tăng trưởng dữ liệu bằng cách tăng tốc độ xử lý của CPU, dung lượng bộ nhớ và tốc độ của đĩa cứng trên máy chủ duy nhất đó, tức là bằng cách mở rộng theo chiều dọc. Hệ quản trị cơ sở dữ liệu quan hệ vẫn còn phù hợp trong môi trường máy tính hiện đại ngày nay, tuy nhiên, những hạn chế của khả năng mở rộng theo chiều dọc là mối quan tâm lớn nhất.

Khối lượng dữ liệu được sử dụng bởi nhiều tổ chức trong những năm gần đây đã lớn hơn dung lượng của một máy chủ duy nhất, do sự bùng nổ của web. Do đó, các công nghệ và kỹ thuật mới đã được phát triển để giải quyết những vấn đề của dữ liệu. Khả năng mở rộng theo chiều ngang đã trở thành trọng tâm mới cho các hệ thống lưu trữ dữ liệu; cung cấp khả năng truyền dữ liệu từ nhiều máy chủ. Các hệ thống lưu trữ dữ liệu mới ra đời để đáp ứng nhu cầu trên, các hệ thống này thường được gọi là kho dữ liệu NoSQL nghĩa là Non-SQL (phi quan hệ) vì chúng không sử dụng ngôn ngữ truy vấn có cấu trúc (SQL) hoặc có mô hình quan hệ.

## 2. CƠ SỞ DỮ LIỆU NOSQL

### 2.1. Tổng quan

NoSQL là thuật ngữ chung cho các hệ cơ sở dữ liệu không sử dụng mô hình dữ liệu quan hệ.

Thuật ngữ này được sử dụng lần đầu bởi Carlo Strozzi vào năm 1998 để đặt tên cho cơ sở dữ liệu Strozzi NoSQL của ông – một cơ sở dữ liệu quan hệ mã nguồn mở nhanh, nhẹ và không liên quan đến SQL.

Năm 2009, Eric Evans đã giới thiệu lại thuật ngữ NoSQL trong một sự kiện thảo luận về "Các cơ sở dữ liệu phân tán, không quan hệ mã nguồn mở" tổ chức bởi Johan Oskarsson của Last.fm. Chính thức đánh dấu bước phát triển mới cho thể hệ cơ sở dữ liệu: phân tán (distributed) và phi quan hệ (non-relational)

### 2.2. Đặc điểm

NoSQL đặc biệt nhấn mạnh đến mô hình lưu trữ cặp giá trị - khóa và hệ thống lưu trữ phân tán:

- Phi quan hệ (Non-relational): relational là thuật ngữ sử dụng đến các mối quan hệ giữa các bảng trong cơ sở dữ liệu quan hệ (Relational Database Management System) sử dụng mô hình gồm 2 loại khóa: khóa chính (primary key) và khóa phụ (foreign key) để ràng buộc dữ liệu nhằm thể hiện tính nhất quán dữ liệu từ các bảng khác nhau. Non-relational là khái niệm không sử dụng các ràng buộc dữ liệu cho tính nhất quán dữ liệu.
- Lưu trữ dữ liệu phân tán.
- Triển khai đơn giản, dễ nâng cấp và mở rộng.
- Mô hình dữ liệu và truy vấn linh hoạt.

## 2.3. Tính năng

### 2.3.1. ACID free

Hiệu suất và khả năng mở rộng tốt hơn trong NoSQL đạt được bằng cách hy sinh khả năng tương thích ACID.

ACID là viết tắt của Atomicity, Consistency, Isolation, Durability (tính nguyên tử, nhất quán, độc lập và bền vững). Về cơ bản, khái niệm ACID xuất phát từ môi trường SQL nhưng do yếu tố nhất quán, các giải pháp NoSQL tránh sử dụng khái niệm ACID. Hệ quản trị cơ sở dữ liệu NoSQL dựa trên các hệ thống phân tán và dữ liệu được lan truyền đến các máy khác nhau trong cụm và nó được yêu cầu để duy trì tính nhất quán.

Ví dụ, nếu có sự thay đổi trong một bảng, thì bắt buộc phải thực hiện thay đổi trong tất cả các máy có dữ liệu. Có thể đạt được sự nhất quán nếu thông tin về quá trình cập nhật lan truyền ngay lập tức qua toàn bộ hệ thống, nếu không, sự không nhất quán được thực hiện và theo cách này, khái niệm ACID tạo ra rắc rối cho các giải pháp NoSQL.

### 2.3.2. BASE

BASE là đảo ngược của ACID và thuật ngữ này là viết tắt của (Basically Available, Soft State, Eventual Consistency) trạng thái Cơ bản sẵn sàng, Mềm mỏng và tính nhất quán cuối cùng. Sử dụng sao chép và phân bổ để giảm khả năng dữ liệu không có sẵn. Do đó, hệ thống luôn sẵn sàng ngay cả khi các mạng con của dữ liệu bị hỏng và không khả dụng trong một khoảng thời gian ngắn. Do đó, nói chung tính khả dụng của BASE đạt được thông qua việc hỗ trợ sự cố từng phần mà không có sự cố toàn bộ hệ thống.

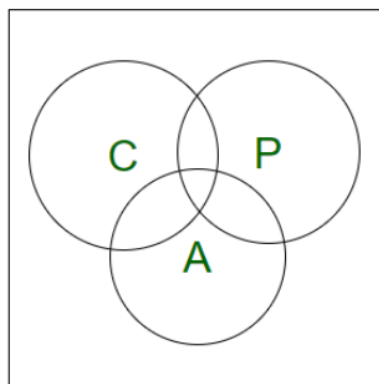
Ví dụ: nếu chúng ta thảo luận về hệ quản trị cơ sở dữ liệu ngân hàng trong các ngân hàng và hai người cố gắng truy cập vào cùng một tài khoản ngân hàng từ hai địa điểm khác nhau thì không chỉ cần cập nhật dữ liệu kịp thời mà còn yêu cầu một số hệ quản trị cơ sở dữ liệu thời gian thực. Một số ví dụ khác về tình huống tương tự có thể là đặt vé trực tuyến và các nền tảng mua sắm trực tuyến.

ACID	BASE
<ul style="list-style-type: none"> <li>• Tính nhất quán cao cho mức ưu tiên cao nhất của giao dịch</li> <li>• Tính khả dụng ít quan trọng hơn</li> <li>• Bi quan (pessimistic)</li> <li>• Phân tích chặt chẽ</li> <li>• Cơ chế phức tạp</li> </ul>	<ul style="list-style-type: none"> <li>• Mức độ sẵn có và mức ưu tiên cao nhất mở rộng</li> <li>• Tính nhất quán yếu</li> <li>• Lạc quan (optimistic)</li> <li>• Hiệu suất tốt</li> <li>• Đơn giản và nhanh chóng</li> </ul>

### 2.3.3. CAP

CAP là viết tắt của Consistency, Availability, Partition tolerance (Tính nhất quán, Tính khả dụng và Dung sai phân vùng). Định lý CAP dựa trên ba nguyên tắc này.

- C viết tắt “Consistency”. Tính nhất quán có nghĩa dữ liệu phải nhất quán và có sẵn trên tất cả các máy phải giống nhau về mọi mặt và quá trình cập nhật phải chạy trên tất cả các máy thường xuyên.
- A viết tắt “Availability”. Tính khả dụng nghĩa là dữ liệu phải luôn sẵn sàng cho khách hàng và phải có thể truy cập bất cứ lúc nào.
- P viết tắt “Partition tolerance”. Dung sai phân vùng nếu do lỗi hệ thống và lỗi trong các node, các hệ quản trị cơ sở dữ liệu phải hoạt động tốt bất chấp các phân vùng mạng vật lý, tức là dung sai phân vùng.



## 2.4. Phân loại

Vì mã nguồn mở, vì vậy có rất nhiều cơ sở dữ liệu ra đời, với những tính năng và đặc điểm riêng. Tuy nhiên, chúng ta có thể phân thành bốn loại chính:

- Key-values Stores
- Column Family Stores
- Document store Databases
- Graph Databases

Ngoài ra có một số cơ sở dữ liệu lai nhiều trong 4 loại trên.

### 2.4.1. Key-values Stores

Key-values Stores là hệ quản trị cơ sở dữ liệu NoSQL đơn giản nhất. Hầu hết Khóa/Giá trị thường có những API sau

```
void Put(string key, byte[] data);  
  
byte[] Get(string key);  
  
void Remove(string key);
```

Ví dụ

Khóa	Giá trị
ho	Nguyễn
ten	Nhật
diachi	Thành phố Hồ Chí Minh

Với khóa-giá trị thì việc truy xuất, xóa, cập nhật giá trị thực đều thông qua khóa tương ứng. Giá trị được lưu dưới dạng BLOB (Binary large object). Xây dựng một khóa/giá trị rất đơn giản và mở rộng chúng cũng rất dễ dàng. Khóa/giá trị có hiệu suất rất tốt bởi vì mô hình truy cập dữ liệu trong khóa/giá trị được tối ưu hóa tối đa. Khóa/giá trị là cơ sở cho tất cả



những loại cơ sở dữ liệu NoSQL khác. Khóa/giá trị rất hữu ích khi cần truy cập dữ liệu theo khóa. Ví dụ như khi cần lưu trữ thông tin phiên giao dịch hoặc thông tin giỏ hàng của người dùng thì khóa/giá trị là một sự lựa chọn hợp lý bởi vì nhờ vào id của người dùng nó có thể nhanh chóng lấy được các thông tin liên quan trong phiên giao dịch hoặc giỏ hàng của người dùng đó.

Một số loại khóa/giá trị phổ biến: Redis, DynamoDB, Azure Table Storage, Riak, Aerospike, KeyDB, ...

#### 2.4.2. Column Family Stores

Loại cơ sở dữ liệu cột là hệ quản trị cơ sở dữ liệu phân tán cho phép truy xuất ngẫu nhiên/tức thời với khả năng lưu trữ một lượng cực lớn dữ liệu có cấu trúc. Dữ liệu có thể tồn tại dạng bảng với hàng tỷ bảng ghi và mỗi bảng ghi có thể chứa hàng triệu cột. Một triển khai từ vài trăm cho tới hàng nghìn node dẫn đến khả năng lưu trữ hàng Petabytes dữ liệu nhưng vẫn đảm bảo hiệu suất cao.

Nhìn bên ngoài vào nó giống với hệ quản trị cơ sở dữ liệu quan hệ nhưng thực sự thì có sự khác biệt rất lớn từ bên trong. Một trong những khác biệt đó chính là việc lưu trữ dữ liệu theo cột so với việc lưu trữ dữ liệu theo dòng (trong hệ quản trị cơ sở dữ liệu quan hệ). Sự khác biệt lớn nhất chính là bản chất của nó. Chúng ta không thể áp dụng cùng một giải pháp mà chúng ta sử dụng trong hệ quản trị cơ sở dữ liệu quan hệ vào trong loại cơ sở dữ liệu họ cột. Đó là bởi vì dữ liệu họ cột là phi quan hệ. Các khái niệm sau đây rất quan trọng để hiểu được hệ quản trị cơ sở dữ liệu column family làm việc như thế nào:

- Column family (cột quan hệ)
- Super column (siêu cột)
- Column (cột)

##### 2.4.2.1. Cột quan hệ

Một cột quan hệ là cách thức dữ liệu được lưu trữ trên đĩa cứng. Tất cả dữ liệu trong một cột sẽ được lưu trên cùng một file. Một họ cột có thể chứa nhiều cột hoặc một cột.

Dòng id	Id 1	Id 2	Id3	...
	Cột giá trị 1	Cột giá trị 2	Cột giá trị 3	
...				

#### 2.4.2.2. Siêu cột

Một siêu cột có thể được dùng như một kiểu từ điển. Nó là một cột có thể chứa những cột khác (mà không phải là siêu cột).

Dòng Id 1	Siêu Id 1			Siêu Id 2			...
	Sub Id 1	Sub Id 2	...	Sub Id 3	Sub Id 4	...	
	Cột giá trị 1	Cột giá trị 2		Cột giá trị 3	Cột giá trị 4		
...							

#### 2.4.2.3. Cột

Một cột là một bộ gồm tên, giá trị và dấu thời gian (thông thường chỉ quan tâm tới khóa-giá trị).

Một số hệ quản trị cơ sở dữ liệu cột phổ biến là: Hbase, Cassandra, Scylla, Hypertable, Apache Flink, ...

#### 2.4.3. Document Store Databases

Khái niệm trung tâm của cơ sở dữ liệu tài liệu là khái niệm “tài liệu”. Về cơ bản thì hệ quản trị cơ sở dữ liệu tài liệu là một khóa-giá trị với giá trị nằm trong một định dạng được biết đến. Mỗi loại cơ sở dữ liệu tài liệu được triển khai khác nhau ở phần cài đặt chi tiết nhưng tất cả tài liệu đều được đóng gói và mã hóa dữ liệu trong một số định dạng tiêu chuẩn hoặc mã hóa. Một số kiểu mã hóa được sử dụng bao gồm XML, YAML, JSON, và BSON, cũng

như kiểu nhị phân như PDF và các tài liệu Microsoft Office (MS Word, Excel, ...). Trên thực tế, tất cả cơ sở dữ liệu tài liệu đều sử dụng JSON(hoặc BSON) hoặc XML.

Các tài liệu bên trong một hệ quản trị cơ sở dữ liệu tài liệu thì tương tự nhau, nó gần giống với khái niệm “bản ghi” hay “dòng” trong hệ quản trị cơ sở dữ liệu quan hệ truyền thống nhưng nó ít cứng nhắc hơn. Tài liệu không bắt buộc phải tuân theo một lược đồ tiêu chuẩn cũng không cần phải có tất cả các thuộc tính, khóa tương tự nhau.

Ví dụ:

Tài liệu 1	Tài liệu 2
<pre>{   ten:"Nga",   diachi:"Đắk Lắk",   sothich:"ngủ" }</pre>	<pre>{   ten:"Mẹ của Nga",   diachi:"Đắk Lắk",   Children:[     {ten:"Nga", Age: 10},     {ten:"Hoàng", Age: 11}   ] }</pre>

Cả hai tài liệu trên có một số thông tin tương tự và một số thông tin khác nhau. Không giống như một cơ sở dữ liệu quan hệ truyền thống, nơi mỗi bản ghi (dòng) có cùng một tập hợp trường dữ liệu và các trường dữ liệu này nếu không sử dụng thì có thể được lưu trữ rỗng, còn trong hệ quản trị cơ sở dữ liệu tài liệu thì không có trường dữ liệu rỗng trong tài liệu. Hệ thống này cho phép thông tin mới được thêm vào mà không cần phải khai báo rõ ràng.

Các tài liệu được đánh dấu trong hệ quản trị cơ sở dữ liệu tài liệu thông qua một khóa duy nhất đại diện cho tài đó. Thông thường, khóa này là một chuỗi đơn giản. Trong một số trường hợp, chuỗi này có thể là một URI hoặc đường dẫn. Chúng ta có thể sử dụng khóa này để lấy tài liệu từ cơ sở dữ liệu. Thông thường, cơ sở dữ liệu vẫn lưu lại một chỉ số trong khóa của tài liệu để tài liệu có thể được tìm kiếm nhanh chóng. Ngoài ra, cơ sở dữ liệu sẽ

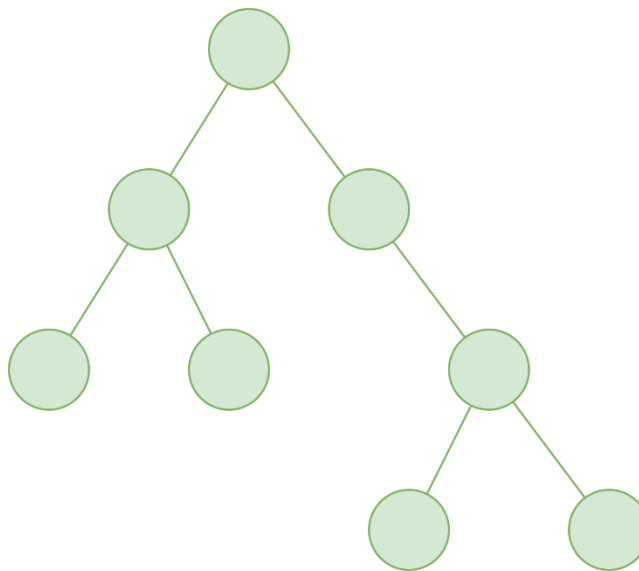
cung cấp một API hoặc ngôn ngữ truy vấn cho phép lấy các tài liệu dựa trên nội dung. Ví dụ, chúng ta muốn truy vấn lấy những tài liệu mà những tài liệu đó có tập trường dữ liệu nhất định với những giá trị nhất định.

Các hệ quản trị cơ sở dữ liệu tài liệu phổ biến là: MongoDB, RethinkDB, RavenDB, Cloud Datastore, No SQL Encryption (Oracle), Elastic, ArangoDB, FaunaDB, OrientDB, gunDB, Cloud Datastore, Azure DocumentDB, ...

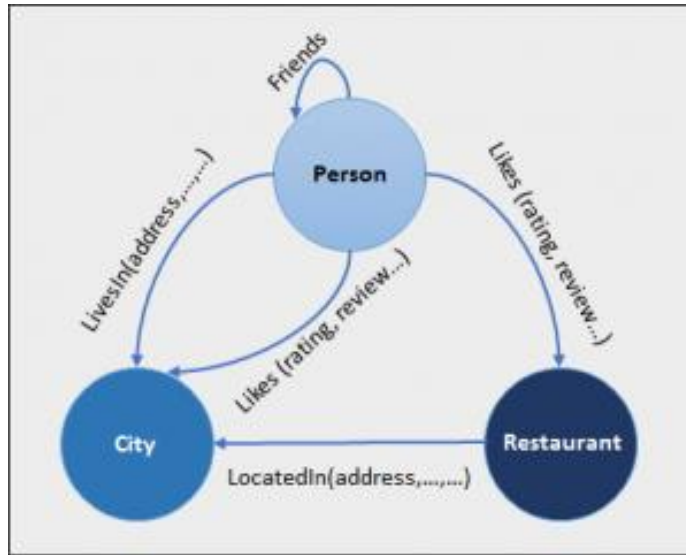
**Lưu ý:** hầu hết cơ sở dữ liệu XML đều là triển khai của hệ quản trị cơ sở dữ liệu tài liệu. Một số XML trong danh sách các phổ biến là: BaseX, eXist, MarkLogic, Sedna.

#### 2.4.4. Graph Databases

Hệ cơ sở dữ liệu đồ thị là một dạng cơ sở dữ liệu được thiết kế riêng cho việc lưu trữ thông tin đồ thị như cạnh, node, các thuộc tính.



Chúng ta có thể nghĩ hệ quản trị cơ sở dữ liệu đồ thị như một hệ quản trị cơ sở dữ liệu tài liệu với các kiểu tài liệu đặc biệt và các mối quan hệ. Một ví dụ điển hình đó chính là mạng xã hội, có thể xem hình bên dưới:



Trong ví dụ trên ta có 3 document và 5 mối quan hệ. Mỗi quan hệ trong cơ sở dữ liệu đồ thị thì có ý nghĩa nhiều hơn con trỏ đơn thuần. Một mối quan hệ có thể một chiều hoặc hai chiều nhưng quan trọng hơn là mỗi quan hệ được phân loại.

Với hệ quản trị cơ sở dữ liệu đồ thị, chúng ta có thể thực hiện các hoạt động đồ thị. Một thao tác cơ bản nhất là điểm giao nhau. Ví dụ như nếu ta muốn biết những người trong cùng một thôn, xã để cùng đi ăn uống thì đơn giản. Nhưng còn bạn bè gián tiếp thì sao, làm sao ta biết được họ. Sử dụng cơ sở dữ liệu đồ thị chúng ta có thể định nghĩa truy vấn sau:

```

new GraphDatabaseQuery
{
    SourceNode = ayende,
    MaxDepth = 3,
    RelationsToFollow = new[]{"đã biết", "Friend", "Ex"},
    Where = node => node.Location == ayende.Location,
    SearchOrder = SearchOrder.BreadthFirst
}.Execute();
  
```

Chúng ta có thể thực hiện những truy vấn phức tạp hơn như lọc trên các thuộc tính quan hệ, xem xét trọng lượng của người đó, ... Cơ sở dữ liệu đồ thị thường được sử dụng để giải

quyết các vấn đề về mạng. Trong thực tế, hầu hết các trang web mạng xã hội đều sử dụng một số hình thức của cơ sở dữ liệu đồ thị để làm những việc mà chúng ta đã biết như: kết bạn, bạn của bạn, ...

Một vấn đề đối với việc mở rộng cơ sở dữ liệu đồ thị là rất khó để tìm thấy một đồ thị con độc lập, có nghĩa là rất khó để ta phân tán graph database thành nhiều mảnh. Có rất nhiều nỗ lực nghiên cứu cho việc này nhưng chưa có bất kỳ giải pháp nào đáng tin cậy được đưa ra.

Một số sản phẩm tiêu biểu của cơ sở dữ liệu đồ thị là: Neo4J, Sones, AllegroGraph, Core Data, DEX, FlockDB, InfoGrid, OpenLink Virtuoso, GraphBase ...

### 3. HỆ QUẢN TRỊ CƠ SỞ DỮ LIỆU REDIS

#### 3.1. Tổng quan về Redis

Câu chuyện bắt đầu khi tác giả của Redis, Salvatore Sanfilippo (nickname: antirez), cố gắng làm những công việc gần như là không thể với SQL Database.

Server của antirez nhận 1 lượng lớn thông tin từ nhiều trang web khác nhau thông qua JavaScript tracker, lưu trữ n page view cho từng trang và hiển thị chúng theo thời gian thực cho user, kèm theo đó là lưu trữ 1 lượng nhỏ lịch sử hiển thị của trang web. Khi số lượng page view tăng đến hàng nghìn page trên 1 giây, antirez không thể tìm ra cách tiếp cận nào thực sự tối ưu cho việc thiết kế database của mình. Tuy nhiên, anh ta nhận ra rằng, việc lưu trữ 1 danh sách bị giới hạn các bản ghi không phải là vấn đề quá khó khăn. Từ đó, ý tưởng lưu trữ thông tin trên RAM và quản lý các page views dưới dạng native data với thời gian pop và push là hằng số được ra đời. Antirez bắt tay vào việc xây dựng prototype bằng C, bổ sung tính năng lưu trữ thông tin trên đĩa cứng và Redis ra đời.

**Redis (REmote DIctionary Server)** là một mã nguồn mở được dùng để lưu trữ dữ liệu có cấu trúc, có thể sử dụng như một database, bộ nhớ cache hay một message broker. Redis hiện nay được ứng dụng rất nhiều, hay sử dụng làm trạm trung chuyển dữ liệu giữa các thành phần trong hệ thống, đặc biệt với những hệ thống đòi hỏi hiệu năng cao.

Redis là cơ sở dữ liệu mang phong cách NoSQL, lưu trữ dữ liệu với dạng KEY-VALUE với nhiều tính năng được sử dụng rộng rãi. Nó có thể hỗ trợ nhiều kiểu dữ liệu như: strings, hashes, lists, sets, sorted sets. Đồng thời có thể cho phép viết kịch bản (scripting) bằng ngôn ngữ Lua.

#### 3.2. Đặc trưng của Redis

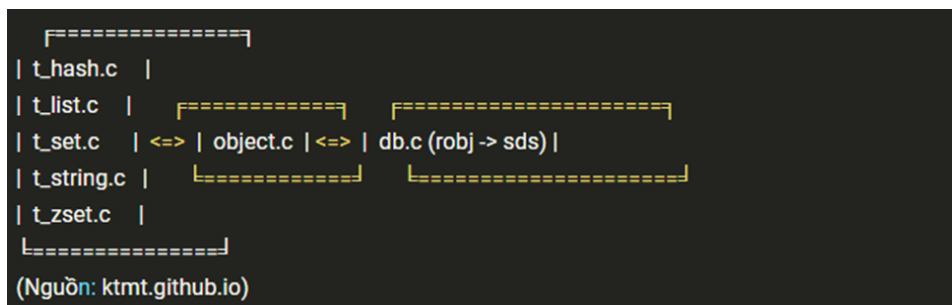
##### 3.2.1. Data Model

Khác với RDMS như MySQL, hay PostgreSQL, Redis không có bảng. Redis lưu trữ data dưới dạng key-value. Thực tế thì memcache cũng làm vậy, nhưng kiểu dữ liệu của memcache bị hạn chế, không đa dạng được như Redis, do đó không hỗ trợ được nhiều thao tác từ phía người dùng. Dưới đây là sơ lược về các kiểu dữ liệu Redis dùng để lưu value.

- **STRINGS:** Có thể là string, integer hoặc float. Redis có thể làm việc với cả string, từng phần của string, cũng như tăng/giảm giá trị của integer, float.
- **LISTS:** Danh sách liên kết của các strings. Redis hỗ trợ các thao tác push, pop từ cả 2 phía của lists, trim dựa theo offset, đọc 1 hoặc nhiều items của lists, tìm kiếm và xóa giá trị.
- **SETS** Tập hợp các string (không được sắp xếp). Redis hỗ trợ các thao tác thêm, đọc, xóa từng phần tử, kiểm tra sự xuất hiện của phần tử trong tập hợp. Ngoài ra Redis còn hỗ trợ các phép toán tập hợp, gồm intersect/union/difference.
- **ZSETS (sorted sets):** Là 1 tập hợp, trong đó mỗi phần tử là map của 1 string (member) và 1 floating-point number (score), danh sách được sắp xếp theo score này. Redis hỗ trợ thao tác thêm, đọc, xóa từng phần tử, lấy ra các phần tử dựa theo range của score hoặc của string.
- **HASHES:** Là kiểu dữ liệu lưu trữ kết hợp giữa các cặp field-value (field-value pairs), ánh xạ giữa các trường (field) và giá trị chuỗi (value). Đây là kiểu dữ liệu khá hoàn hảo để biểu diễn các đối tượng được lưu trữ trong Redis. Redis hỗ trợ các thao tác thêm, đọc, xóa từng phần tử, cũng như đọc tất cả giá trị.

➤ Lưu ý:

Trang web [ktmt.github.io](https://ktmt.github.io) đưa ra loạt bài phân tích về source code Redis (viết bằng C), trong đó có 1 phần về kiểu dữ liệu của Redis. Tham khảo các bài viết đó, chúng ta có thể thấy Redis sử dụng 1 layer mô tả dữ liệu ở mức độ abstract, là redisObjectr-robj (định nghĩa trong redis.h), các thao tác cơ bản của db (db.c) đều làm việc trực tiếp với robj và không cần biết đến sự tồn tại của các kiểu strings, lists, hashes, sets, zsets. Sơ đồ tổ chức có thể tham khảo trong mô hình dưới đây.

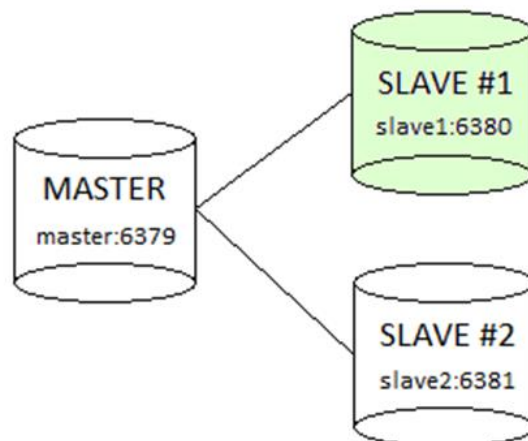




Thiết kế này giúp thao tác làm việc với các kiểu dữ liệu khác nhau trở nên dễ dàng quản lý hơn, đồng thời hỗ trợ việc tăng cường số lượng kiểu dữ liệu trong tương lai.

### 3.2.2. Master/Slave Replication

Redis hỗ trợ nhân bản Master/Slave Replication. Dữ liệu từ bất kỳ máy chủ Redis nào cũng có thể sao chép thành bất kỳ bản sao nào. Đây không phải là đặc trưng quá nổi bật, các DBMS khác đều có tính năng này, tuy nhiên nêu ra ở đây để nhắc nhở rằng, Redis không kém cạnh các DBMS về tính năng Replication.



### 3.2.3. In-memory

Đây là điều gây ấn tượng mạnh nhất khi bắt đầu tìm hiểu về Redis. Không như các DBMS khác lưu trữ dữ liệu trên đĩa cứng, Redis lưu trữ dữ liệu trên RAM, và đương nhiên là thao tác đọc/ghi trên RAM.

### 3.2.4. Redis Persistence

Với người làm công nghệ thông tin bình thường, ai cũng hiểu thao tác trên RAM nhanh hơn nhiều so với trên ổ cứng, nhưng chắc chắn chúng ta sẽ có cùng câu hỏi: Điều gì xảy ra với data của chúng ta khi server bị tắt?

Rõ ràng là toàn bộ dữ liệu trên RAM sẽ bị mất khi tắt server, vậy làm thế nào để Redis bảo toàn data và vẫn duy trì được ưu thế xử lý dữ liệu trên RAM.

Như đã đề cập ở trên, mặc dù làm việc với data dạng key-value lưu trữ trên RAM, Redis vẫn cần lưu trữ dữ liệu trên ổ cứng. Có 2 lý do cho việc này, 1 là để đảm bảo toàn vẹn dữ liệu khi có sự cố xảy ra (server bị tắt nguồn) cũng như tái tạo lại dataset khi restart server, 2 là để gửi data đến các slave server, phục vụ cho tính năng replication. Redis cung cấp 2 phương thức chính cho việc sao lưu dữ liệu ra ổ cứng, đó là RDB và AOF.

#### 3.2.4.1. RDB (Redis DataBase file)

Cách thức làm việc RDB thực hiện tạo và sao lưu snapshot của DB vào ổ cứng sau mỗi khoảng thời gian nhất định.

- Ưu điểm
  - RDB cho phép người dùng lưu các version khác nhau của DB, rất thuận tiện khi có sự cố xảy ra.
  - Bằng việc lưu trữ data vào 1 file cố định, người dùng có thể dễ dàng chuyển data đến các data centers khác nhau, hoặc chuyển đến lưu trữ trên Amazon S3.
  - RDB giúp tối ưu hóa hiệu năng của Redis. Tiến trình Redis chính sẽ chỉ làm các công việc trên RAM, bao gồm các thao tác cơ bản được yêu cầu từ phía client như thêm/đọc/xóa, trong khi đó 1 tiến trình con sẽ đảm nhiệm các thao tác disk I/O. Cách tổ chức này giúp tối đa hiệu năng của Redis.
  - Khi restart server, dùng RDB làm việc với lượng data lớn sẽ có tốc độ cao hơn là dùng AOF.
- Nhược điểm
  - RDB không phải là lựa chọn tốt nếu bạn muốn giảm thiểu tối đa nguy cơ mất mát dữ liệu. Thông thường người dùng sẽ set up để tạo RDB snapshot 5 phút 1 lần (hoặc nhiều hơn). Do vậy, trong trường hợp có sự cố, Redis không thể hoạt động, dữ liệu trong những phút cuối sẽ bị mất.
  - RDB cần dùng *fork()* để tạo tiến trình con phục vụ cho thao tác disk I/O. Trong trường hợp dữ liệu quá lớn, quá trình *fork()* có thể tốn thời gian và

server sẽ không thể đáp ứng được request từ client trong vài milisecond hoặc thậm chí là 1 second tùy thuộc vào lượng data và hiệu năng CPU.

### 3.2.4.2. AOF (Append Only File)

Cách thức làm việc: AOF lưu lại tất cả các thao tác write mà server nhận được, các thao tác này sẽ được chạy lại khi restart server hoặc tái thiết lập dataset ban đầu.

- Ưu điểm
  - Sử dụng AOF sẽ giúp đảm bảo dataset được bền vững hơn so với dùng RDB. Người dùng có thể config để Redis ghi log theo từng câu query hoặc mỗi giây 1 lần.
  - Redis ghi log AOF theo kiểu thêm vào cuối file sẵn có, do đó tiến trình seek trên file có sẵn là không cần thiết. Ngoài ra, kể cả khi chỉ 1 nửa câu lệnh được ghi trong file log (có thể do ổ đĩa bị full), Redis vẫn có cơ chế quản lý và sửa chữa lỗi đó (redis-check-aof).
  - Redis cung cấp tiến trình chạy nền, cho phép ghi lại file AOF khi dung lượng file quá lớn. Trong khi server vẫn thực hiện thao tác trên file cũ, 1 file hoàn toàn mới được tạo ra với số lượng tối thiểu operation phục vụ cho việc tạo dataset hiện tại. Và 1 khi file mới được ghi xong, Redis sẽ chuyển sang thực hiện thao tác ghi log trên file mới.
- Nhược điểm
  - File AOF thường lớn hơn file RDB với cùng 1 dataset.
  - AOF có thể chậm hơn RDB tùy theo cách thức thiết lập khoảng thời gian cho việc sao lưu vào ổ cứng. Tuy nhiên, nếu thiết lập log 1 giây 1 lần có thể đạt hiệu năng tương đương với RDB.
  - Developer của Redis đã từng gặp phải bug với AOF (mặc dù là rất hiếm), đó là lỗi AOF không thể tái tạo lại chính xác dataset khi restart Redis. Lỗi này chưa gặp phải khi làm việc với RDB bao giờ.

### 3.2.4.3. Nên dùng RDB hay AOF

Mỗi phương thức đều có ưu/nhược điểm riêng, và có lẽ cần nhiều thời gian làm việc với Redis cũng như tùy theo ứng dụng mà đưa ra lựa chọn thích hợp. Nhiều người chọn AOF bởi nó đảm bảo toàn vẹn dữ liệu rất tốt, nhưng Redis developer lại khuyến cáo nên dùng cả RDB, bởi nó rất thuận tiện cho việc backup database, tăng tốc độ cho quá trình khởi động lại cũng như tránh được lỗi của AOF.

Cũng cần lưu ý thêm rằng, Redis cho phép không sử dụng tính năng lưu trữ thông tin trong ổ cứng (không RDB, không AOF), đồng thời cũng cho phép dùng cả 2 tính năng này trên cùng 1 instance. Tuy nhiên khi restart server, Redis sẽ dùng AOF cho việc tái tạo dataset ban đầu, bởi AOF sẽ đảm bảo không bị mất mát dữ liệu tốt hơn là RDB.

## 3.3. Cấu trúc dữ liệu của Redis

### 3.3.1. Strings

Là loại giá trị cơ bản nhất của Redis, giá trị của string có thể đạt tới 512MB.

#### Command:

- SET – MSET: set giá trị cho một (SET) hoặc nhiều khóa (MSET), thay đổi giá trị cũ thành mới nếu trùng

```
127.0.0.1:6379> set name1 Huy _
OK
127.0.0.1:6379> mset name2 Nga name3 Hoan_
OK
```

- SETNX – MSETNX: set giá trị cho một (SETNX) hoặc nhiều khóa (MSETNX), không thay đổi giá trị cũ thành mới, nếu trùng khóa thì sẽ không hoạt động

```
127.0.0.1:6379> setnx name1 Nga_
(integer) 0
127.0.0.1:6379> msetnx name1 Nga name2 Huy_
(integer) 0
127.0.0.1:6379> msetnx class PhanTan school UIT
(integer) 1
```

```
127.0.0.1:6379> mget class school_
1) "PhanTan"
2) "UIT"
127.0.0.1:6379> mget name1 name2 name3
1) "Huy"
2) "Nga"
3) "Hoan"
```

- GET – MGET: Lấy giá trị một (GET) hoặc nhiều (MGET) khóa, nếu không có giá trị thì trả về nil

```
127.0.0.1:6379> get name2_
"Nga"
127.0.0.1:6379> mget name1 name3
1) "Huy"
2) "Hoan"
127.0.0.1:6379> mget name4
1) (nil)
127.0.0.1:6379>
```

- APPEND: Thêm giá trị vào cuối chuỗi

```
127.0.0.1:6379> set name Huy_
OK
127.0.0.1:6379> get name_
"Huy"
127.0.0.1:6379> append name HTCL
(integer) 7
127.0.0.1:6379> get name
"HuyHTCL"
```

- GETRANGE: Trả về giá trị trong chuỗi trong khoảng được chọn

```
127.0.0.1:6379> getrange name 0 -1
"HuyHTCL"
127.0.0.1:6379> getrange name 0 2
"Huy"
127.0.0.1:6379> getrange name 0 4
"HuyHT"
127.0.0.1:6379>
```

- RENAME: Đổi tên key nếu không có sẽ tự thêm

```
127.0.0.1:6379> set name Huy
OK
127.0.0.1:6379> get name_
"Huy"
127.0.0.1:6379> rename name myname_
OK
127.0.0.1:6379> get myname_
"Huy"
```

- RENAMENX: Đổi tên key nếu không có sẽ báo thất bại

```
127.0.0.1:6379> renamenx myname myname1_
(integer) 1
127.0.0.1:6379> get myname1
"Huy"
127.0.0.1:6379> renamenx ten hoten
(error) ERR no such key
```

- GETSET: set và trả về giá trị key

```
127.0.0.1:6379> getset gv "MinhNhut"
"MinhNhut"
127.0.0.1:6379> get gv
"MinhNhut"
```

- SETEX: set giá trị tồn tại trong một khoảng thời gian nhất định (timeout)

```
127.0.0.1:6379> setex chao 10 Hello
OK
127.0.0.1:6379> ttl chao_
(integer) 6
127.0.0.1:6379> ttl chao
(integer) 4
127.0.0.1:6379> ttl chao_
(integer) 2
127.0.0.1:6379> ttl chao
(integer) -2
127.0.0.1:6379> get chao_
(nil)
127.0.0.1:6379>
```

- PERSIST: loại bỏ timeout

```
127.0.0.1:6379> setex chao 10 Hello_
OK
127.0.0.1:6379> ttl chao_
(integer) 7
127.0.0.1:6379> persist chao_
(integer) 1
127.0.0.1:6379> ttl chao_
(integer) -1
127.0.0.1:6379> get chao
"Hello"
127.0.0.1:6379>
```

- SCAN: Lặp lại key trong cơ sở dữ liệu. Chỉ trả về một phần nhỏ. Lấy con trỏ/ vị trí làm tham số. Máy chủ trả về con trỏ được cập nhật với mỗi lần gọi lệnh. Điều này có thể được sử dụng trong đối số của lần gọi tiếp theo. Bắt đầu lặp lại khi con trỏ được đặt thành 0. Chấm dứt khi con trỏ trở về từ máy chủ là 0

**Đảm bảo:** Lặp lại đầy đủ để truy xuất tất cả các phần tử đầu đến cuối. Không trả về bất kỳ phần tử nào không có trong dữ liệu nhập vào từ đầu đến cuối

```
127.0.0.1:6379> scan 0
1) "3"
2) 1) "key5"
   2) "num"
   3) "gv"
   4) "key4"
   5) "key1"
   6) "key2"
   7) "name3"
   8) "key3"
   9) "chao"
  10) "name2"
  11) "school"
127.0.0.1:6379> scan 4
1) "7"
2) 1) "gv"
   2) "key4"
   3) "key1"
   4) "key2"
   5) "name3"
   6) "key3"
   7) "chao"
   8) "name2"
   9) "school"
  10) "myname1"
  11) "name1"
  12) "list"
  13) "class"
127.0.0.1:6379> scan 15
1) "0"
2) (empty list or set)
127.0.0.1:6379>
```

- COUNT: Có thể được xác định trong SCAN để ghi đè lên giá trị mặc định được trả về mỗi lần lặp. Người dùng có thể chỉ định khối lượng công việc được thực hiện trong mỗi lần gọi. Mặc định COUNT là 10. COUNT có thể được thay đổi từ 1 lần lặp lại tiếp theo

```
127.0.0.1:6379> scan 0 count 5
1) "14"
2) 1) "key5"
   2) "num"
   3) "gv"
   4) "key4"
   5) "key1"
```

- MATCH: Lặp lại các phần tử phù hợp với một mẫu được chỉ định

```
127.0.0.1:6379> scan 10 match key*_
1) "7"
2) 1) "key4"
   2) "key1"
   3) "key2"
   4) "key3"
127.0.0.1:6379> scan 10 match ten*_
1) "7"
2) (empty list or set)
```

### 3.3.2. Lists

Là một danh sách nhóm hoặc danh sách chuỗi, được sắp xếp theo thuật toán Insertion Sort.

#### Command:

- LPUSH: chèn phần tử vào đầu danh sách

```
127.0.0.1:6379> lrange list 0 -1
1) "A"
2) "B"

127.0.0.1:6379> lpush list start
(integer) 3
127.0.0.1:6379> lrange list 0 -1
1) "start"
2) "A"
3) "B"
```



- RPUSH: chèn phần tử vào cuối danh sách

```
127.0.0.1:6379> rpush list A_
(integer) 1
127.0.0.1:6379> rpush list B
(integer) 2

127.0.0.1:6379> lrange list 0 -1
1) "A"
2) "B"
```

- LINSERT: chèn phần tử vào vị trí cạnh phần tử đã có

```
127.0.0.1:6379> linsert list before B X
(integer) 4
127.0.0.1:6379> lrange list 0 -1
1) "A"
2) "X"
3) "B"
4) "C"
127.0.0.1:6379>

127.0.0.1:6379> linsert list after B Y
(integer) 5
127.0.0.1:6379> lrange list 0 -1
1) "A"
2) "X"
3) "B"
4) "Y"
5) "C"
```

- LRANGE: Trả về các phần tử theo vị trí chỉ định

```
127.0.0.1:6379> lrange list 0 -1
1) "A"
2) "X"
3) "B"
4) "Y"
5) "C"
127.0.0.1:6379> lrange list 0 2
1) "A"
2) "X"
3) "B"
127.0.0.1:6379>
```

- LLEN: Số phần tử trong danh sách

```
127.0.0.1:6379> lrange list 0 -1_
1) "A"
2) "X"
3) "B"
4) "Y"
5) "C"
127.0.0.1:6379> llen list
(integer) 5
127.0.0.1:6379>
```

- LPOP: Xóa phần tử đầu danh sách

```
127.0.0.1:6379> lpop list_
"A"
127.0.0.1:6379> lrange list 0 -1_
1) "X"
2) "B"
3) "Y"
4) "C"
127.0.0.1:6379>
```

- RPOP: Xóa phần tử cuối danh sách

```
127.0.0.1:6379> lrange list 0 -1_
1) "X"
2) "B"
3) "Y"
127.0.0.1:6379>
```

### 3.3.3. Sets

Là một tập hợp ngẫu nhiên các chuỗi giá trị không trùng nhau

#### Command:

- SADD: Thêm một giá trị vào SET, bỏ qua nếu trùng

```
127.0.0.1:6379> sadd name Huy_
(integer) 1
127.0.0.1:6379> sadd name Huy_
(integer) 0
127.0.0.1:6379>
```

- SREM: Xóa một giá trị từ SET

```
127.0.0.1:6379> srem name Huy_
(integer) 1
127.0.0.1:6379>
```

- SISMEMBER: Kiểm tra sự tồn tại của giá trị trong SET

```
127.0.0.1:6379> sadd name Huy_
(integer) 1
127.0.0.1:6379> sismember name Huy_
(integer) 1
127.0.0.1:6379> sismember name Nga_
(integer) 0
```

- SMEMBERS: Tìm toàn bộ phần tử trong SET

```
127.0.0.1:6379> smembers name
1) "Hoan"
2) "Huy"
3) "Nga"
```

- SCARD: Đếm số phần tử trong SET

```
127.0.0.1:6379> smembers name
1) "Hoan"
2) "Huy"
3) "Nga"
127.0.0.1:6379> scard name
(integer) 3
```

- SMOVE: Chuyển vị trí của giá trị giữa 2 SET

```
127.0.0.1:6379> sadd ten Nhut_
(integer) 1
127.0.0.1:6379> smove name ten Nga_
(integer) 1
127.0.0.1:6379> smembers name_
1) "Hoan"
2) "Huy"
127.0.0.1:6379> smembers ten_
1) "Nhut"
2) "Nga"
```

- SUNION: Hợp các SET

```
127.0.0.1:6379> sunion name ten
1) "Huy"
2) "Hoan"
3) "Nhut"
4) "Nga"
```

- SDIFF: Trả về giá trị không cùng xuất hiện trong các SET

```
127.0.0.1:6379> sadd ten Huy
(integer) 1
127.0.0.1:6379> sdiff name ten_
1) "Hoan"
```

- SRANDMEMBER: Trả về giá trị ngẫu nhiên trong SET

```
127.0.0.1:6379> srandmember ten_
"Nga"
127.0.0.1:6379> srandmember ten
"Nhut"
127.0.0.1:6379>
```

- SPOP: Xóa giá trị ngẫu nhiên trong SET

```
127.0.0.1:6379> spop ten_
"Huy"
127.0.0.1:6379> smembers ten_
1) "Nhut"
2) "Nga"
```

### 3.3.4. Sorted Sets

Tương tự Sets ở việc lưu các giá trị không trùng nhau. Điểm khác nhau giữa hai kiểu dữ liệu này là mỗi giá trị trong Sorted Sets gắn liền với một giá trị điểm số (score) dùng để sắp xếp các giá trị trong Sorted Sets theo thứ tự từ phần tử có điểm số thấp nhất đến cao nhất.

Score: Là giá trị bắt buộc phải có khi tạo một phần tử trong Sorted Sets, phải là giá trị số kiểu float. Các phần tử có thể có cùng giá trị Score.

#### Command:

- ZADD: Thêm một hay nhiều phần tử vào Sorted Set hoặc cập nhật lại giá trị score của phần tử nếu phần tử đó đã tồn tại

```
127.0.0.1:6379> zadd hoten 100 "Quang Huy"
(integer) 1
127.0.0.1:6379> zadd hoten 200 "Thuy Nga"
(integer) 1
127.0.0.1:6379> zadd hoten 300 "Trong Hoan" 400 "Minh Nhut"
(integer) 2
127.0.0.1:6379>
```

- ZREM: Xóa một hay nhiều phần tử từ Sorted Set. Phần tử không tồn tại sẽ được bỏ qua. Lỗi sẽ được trả về trong trường hợp key là tồn tại và dữ liệu của key không phải là một Sorted Set

```
127.0.0.1:6379> zrem hoten "Thuy Nga"
(integer) 1
127.0.0.1:6379>
```

- ZRANGE: Danh sách sắp xếp theo thứ tự thấp nhất đến cao nhất

```
127.0.0.1:6379> zrange hoten 0 -1
1) "Quang Huy"
2) "Trong Hoan"
3) "Minh Nhut"
127.0.0.1:6379> zrange hoten 0 1
1) "Quang Huy"
2) "Trong Hoan"
127.0.0.1:6379>
```

- ZREVRANGE: Danh sách sắp xếp theo thứ tự cao nhất đến thấp nhất

```
127.0.0.1:6379> zrevrange hoten 0 -1
1) "Minh Nhut"
2) "Trong Hoan"
3) "Quang Huy"
127.0.0.1:6379> zrevrange hoten 0 1
1) "Minh Nhut"
2) "Trong Hoan"
127.0.0.1:6379>
```

- ZRANGEBYSCORE: Danh sách với giá trị score nằm trong khoảng từ MIN đến MAX (bao gồm cả các giá trị MIN và MAX). Sắp xếp theo thứ tự score thấp nhất đến cao nhất

```
127.0.0.1:6379> zrangebyscore hoten 100 200
1) "Quang Huy"
127.0.0.1:6379> zrangebyscore hoten 100 300
1) "Quang Huy"
2) "Trong Hoan"
127.0.0.1:6379> zrangebyscore hoten 1000 3000
(empty list or set)
```

- ZRANK: Trả về thứ hạng (rank) của phần tử trong Sorted Set. Sắp xếp theo thứ tự score nhỏ nhất đến score là lớn nhất. Giá trị thứ hạng bắt đầu từ 0 có nghĩa là phần tử có giá trị score nhỏ nhất sẽ có thứ hạng là 0

```
127.0.0.1:6379> zrank hoten "Quang Huy"
(integer) 0
127.0.0.1:6379> zrank hoten "Minh Nhat"
(integer) 2
```

- ZCARD: Đếm số phần tử trong Sorted Set

```
127.0.0.1:6379> zcard hoten
(integer) 3
```

- ZCOUNT: Đếm số phần tử trong khoảng MIN – MAX

```
127.0.0.1:6379> zcount hoten 100 200_
(integer) 1
127.0.0.1:6379> zcount hoten 1000 2000_
(integer) 0
```

- ZSCORE: Trả về giá trị score của phần tử nằm trong Sorted Set

```
127.0.0.1:6379> zscore hoten "Quang Huy"
"100"
```

- ZINCRBY: Tăng giá trị score của phần tử nằm trong Sorted Set. Nếu phần tử không tồn tại thì sẽ thêm mới phần tử đó vào Sorted Set với giá trị score là giá trị INCREMENT

```
127.0.0.1:6379> zincrby hoten 10 "Quang Huy"
"110"
```

```
127.0.0.1:6379> zincrby hoten 10 "Khanh An"
"10"
```

### 3.3.5. Hashes

Là kiểu dữ liệu lưu trữ kết hợp giữa các cặp field-value (field-value pairs). Đây là kiểu dữ liệu khá hoàn hảo để biểu diễn các đối tượng được lưu trữ trong Redis.

### Command:

- HSET – HMSET: Thêm một (HSET) hoặc nhiều (HMSET) field – value vào Hash hoặc cập nhật lại giá trị của field nếu đã tồn tại

```
127.0.0.1:6379> hset huy name "Quang Huy"  
(integer) 1
```

```
127.0.0.1:6379> hmset nga name "Thuy Nga" age "20" email "nga@gmail.com"  
OK
```

- HGET – HMGET – HGETALL: Lấy giá trị của một (HGET), nhiều (HMGET) hoặc tất cả (HGETALL) field trong Hash được chỉ định

```
127.0.0.1:6379> hget nga name  
"Thuy Nga"  
127.0.0.1:6379> hmget nga name age email  
1) "Thuy Nga"  
2) "20"  
3) "nga@gmail.com"  
127.0.0.1:6379> hgetall nga  
1) "name"  
2) "Thuy Nga"  
3) "age"  
4) "20"  
5) "email"  
6) "nga@gmail.com"
```

- HDEL: Xóa các field được chỉ định trong Hash. Các field không tồn tại trong Hash sẽ được bỏ qua

```
127.0.0.1:6379> hdel nga age class  
(integer) 1  
127.0.0.1:6379> hgetall nga  
1) "name"  
2) "Thuy Nga"  
3) "email"  
4) "nga@gmail.com"
```

- HEXISTS: Kiểm tra field được chỉ định có tồn tại trong Hash hay không

```
127.0.0.1:6379> hexists nga school  
(integer) 0  
127.0.0.1:6379> hexists nga name  
(integer) 1
```

- HKEYS: Lấy ra tất cả các field trong Hash được chỉ định

```
127.0.0.1:6379> hkeys nga
1) "name"
2) "email"
```

- HLEN: Cho biết số lượng field có trong Hash được chỉ định

```
127.0.0.1:6379> hset nga class "HTCL"
(integer) 1
127.0.0.1:6379> hlen nga_
(integer) 3
127.0.0.1:6379> hdel nga class_
(integer) 1
127.0.0.1:6379> hlen nga_
(integer) 2
```

- HVALS: Lấy ra tất cả các giá trị có trong Hash được chỉ định

```
127.0.0.1:6379> hvals nga_
1) "Thuy Nga"
2) "nga@gmail.com"
127.0.0.1:6379>
```

### 3.4. Cơ chế phân tán

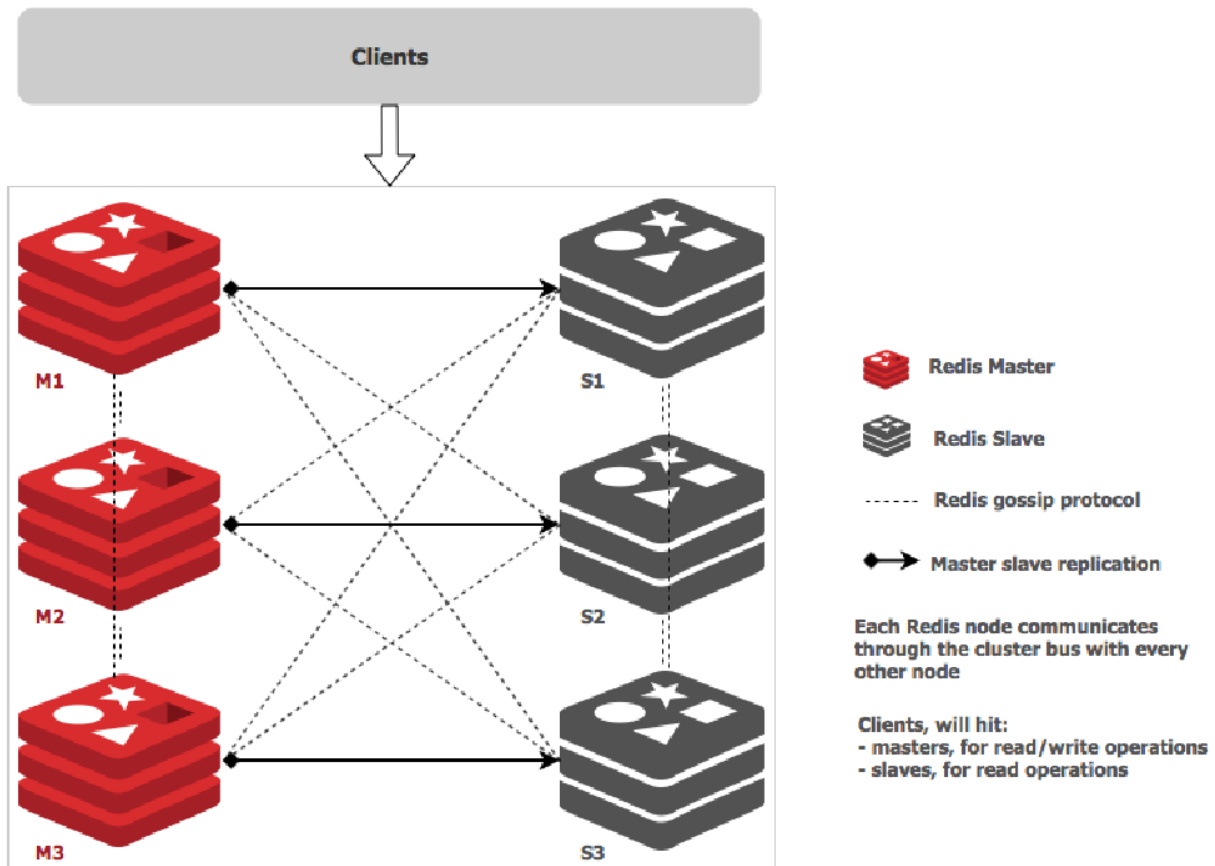
#### 3.4.1. Giao tiếp giữa các node

Các cụm kết nối bằng cách sử dụng một bus TCP và một giao thức nhị phân, được gọi là Redis Cluster Bus. Mọi node kết nối với node khác trong cụm bằng cách sử dụng bus cluster.

Các node sử dụng giao thức gossip để truyền thông tin về cụm nhằm phát hiện ra các node mới, gửi các gói ping để đảm bảo tất cả các node khác đang hoạt động bình thường.

Các cụm không thể yêu cầu proxy, các máy khách có thể được chuyển hướng đến các node khác bằng cách sử dụng lỗi chuyển hướng – MOVED và ASK.





### 3.4.2. Phân vùng dữ liệu

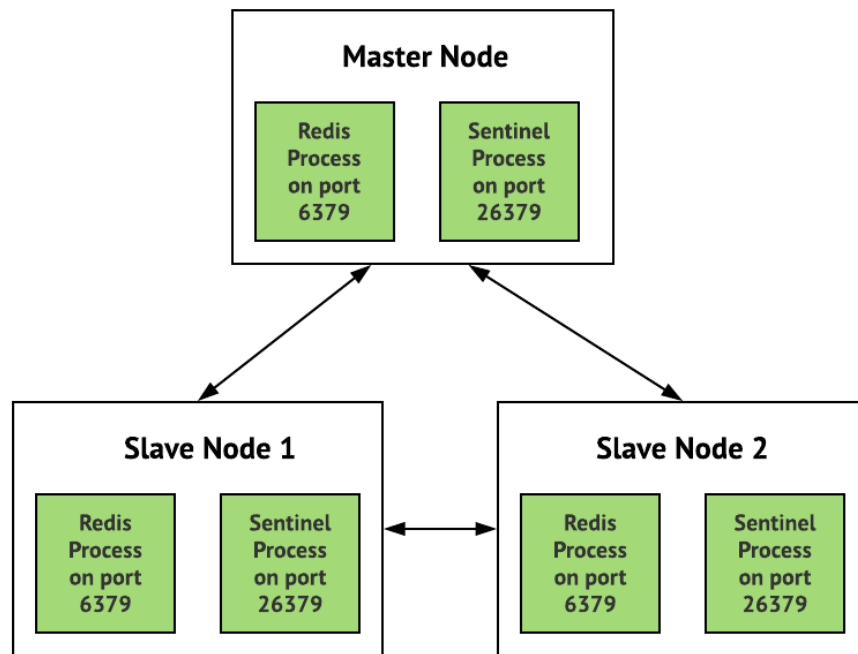
Redis là một công cụ hữu dụng để phát triển các ứng dụng. Có thể dùng Redis để cache, quản lý các session trong ứng dụng một cách nhanh chóng. Redis có tốc độ ghi và đọc dữ liệu nhanh hơn rất nhiều so với các cơ sở dữ liệu khác. Nó cũng xây dựng tính năng pub/sub message borker. Điều này nghĩa là có thể sử dụng Redis để thực hiện việc lắng nghe và phát đi các sự kiện. Ví dụ một hệ thống thông báo subscribe một channel trên redis message broker, nó có thể nhận được các thông báo của các publishers khi có một sự kiện được phát đến channel này. Ngoài ra Redis hỗ trợ nhiều kiểu dữ liệu khác nhau và các phương thức để truy xuất dữ liệu hợp lý.

Redis cũng giống các cơ sở dữ liệu khác, Redis cần cấu hình để đảm bảo tính sẵn sàng và hoạt động của nó với Redis Cluster và Redis Sentinel.

### 3.4.2.1. Redis Sentinel

Được sử dụng khi tốc độ không phải là mối quan tâm của bạn, nó là lựa chọn tuyệt vời để đáp ứng tính sẵn sàng của Redis.

#### Initial Setup



**Configuration: quorum = 2**

### 3.4.2.2. Redis Cluster

Cung cấp tính sẵn sàng cao với giải pháp phân cụm. Nó là lựa chọn tuyệt vời để đảm bảo tính sẵn sàng trong khi vẫn có tốc độ truy vấn nhanh vào dữ liệu.

Khi khởi động một Redis cluster, phải chọn cách để phân chia giữa các node trong cluster. Đây được gọi là chọn cách phân vùng cho cluster.

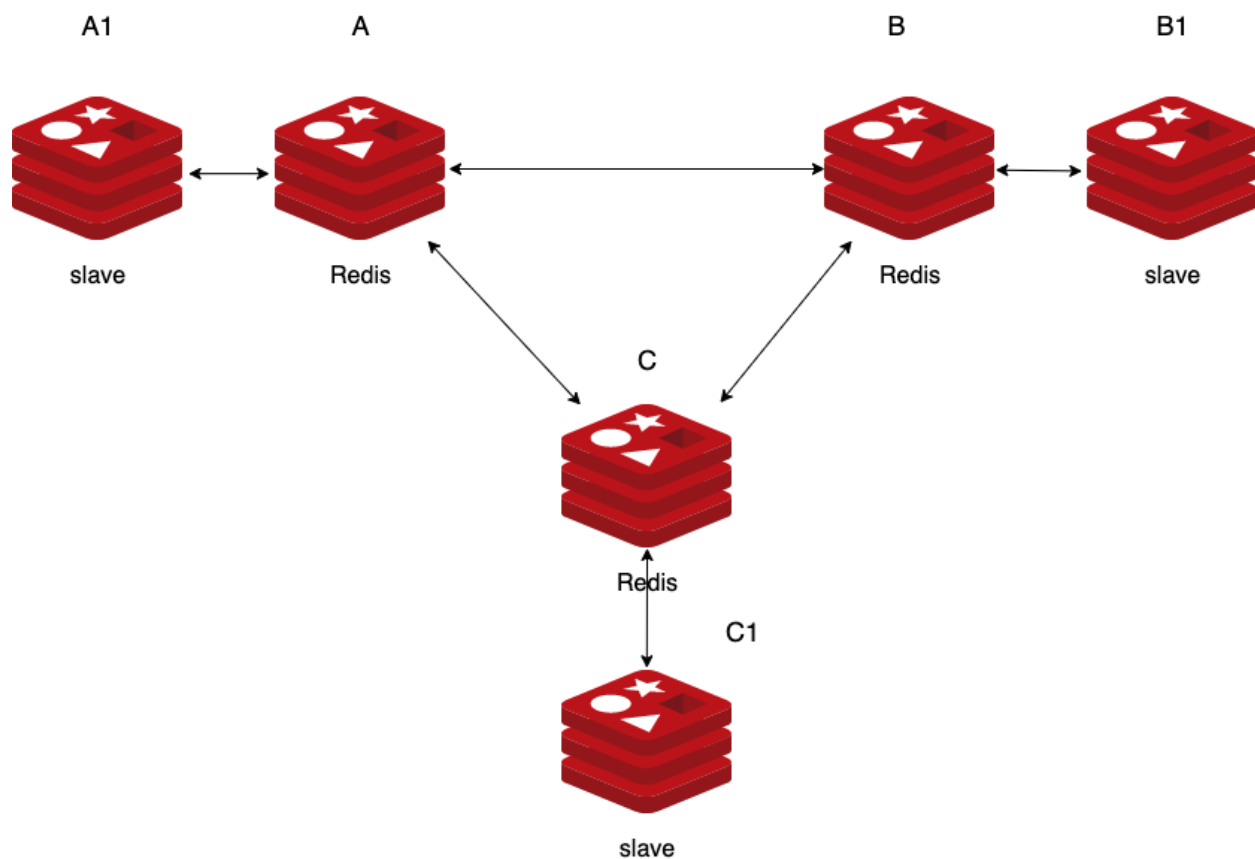
Mỗi node, Redis Cluster yêu cầu mở 2 kết nối TCP. Cổng TCP Redis bình thường được sử dụng sẽ phục vụ cho clients, ví dụ 6379, cộng với cổng thu được bằng cách thêm 10000 vào cổng dữ liệu ta được cổng 16379. Đảm bảo bạn mở cả hai cổng trong tường lửa của mình, nếu không các node cụm Redis sẽ không thể giao tiếp với nhau.

Redis Cluster sử dụng hash slot với 16384 hash slots.

Mỗi node trong cluster sẽ chịu trách nhiệm cho một tập hợp con của các vị trí hash slots, ví dụ nếu chúng ta có cluster với 3 node:

- Node A chứa hash slots từ 0 đến 5500.
- Node B chứa hash slots từ 5501 đến 11000.
- Node C chứa hash slots từ 11001 đến 16383.

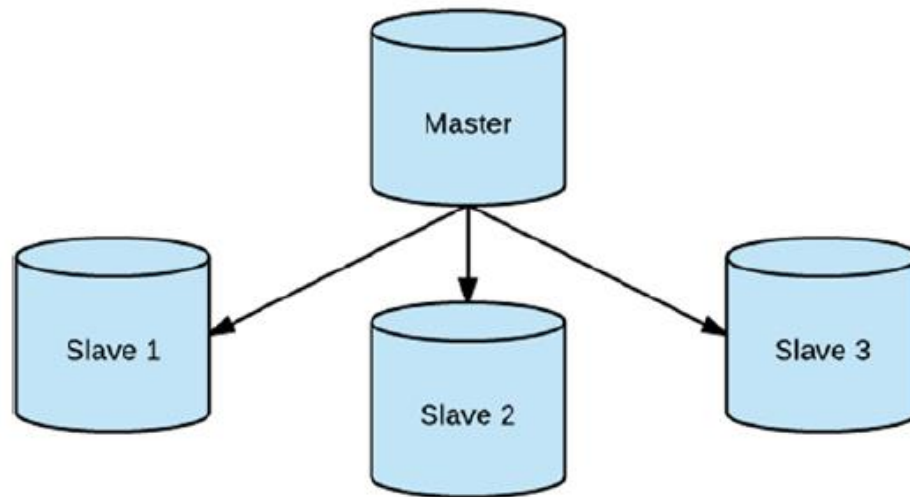
Redis cluster sử dụng cấu hình master-slave để hỗ trợ môi trường phân tán. Redis Cluster hỗ trợ nhiều hoạt động sao cho tất cả các khóa liên quan đến một lệnh thực thi duy nhất (hoặc toàn bộ giao dịch, hoặc thực thi tập lệnh CLI) đều thuộc cùng một khe bấm. Người dùng có thể buộc nhiều khóa trở thành một phần của cùng một khe bấm bằng cách sử dụng một khái niệm gọi là thẻ bấm.



### 3.4.3. Mô hình Master-Slave Replication

Với mô hình của Redis Replication rất dễ dàng để sử dụng và định dạng cấu hình master-slave: nó cho phép sao chép chính xác các trường hợp Redis bản sao của các trường hợp tổng thể. Bản sao sẽ tự động kết nối lại với máy chủ khi liên kết bị hỏng và sẽ cố gắng thành một bản sao chính xác của nó dù có bất kỳ điều gì xảy ra với máy chủ.

Không giống như Cassandra, không thể điều chỉnh mức độ nhất quán. Đó là vì khi có một request ghi dữ liệu tới Redis, node master sẽ ghi dữ liệu lên chính nó trước, và "ngay lập tức" trả thông báo thành công cho máy khách. Sau đó sao chép vào các nodeslave mới thực hiện (quá trình này bất đồng bộ). Điều gì sẽ xảy ra nếu node master bị treo trước khi sao chép dữ liệu đến các node slave. Khi đó, khách hàng đã nhận được thông báo thành công nhưng thực chất là dữ liệu đã mất.



#### 3.4.3.1. Cơ chế hoạt động

Hệ thống hoạt động với 3 cơ chế chính:

- Khi master và slave được kết nối tốt thì bản gốc sẽ giữ bản sao được cập nhật bằng cách gửi 1 dòng luồng lệnh đến bản sao, để sao chép các hiệu ứng hay tập lệnh trên tập dữ liệu xảy ra do bên phía máy chủ : máy khách viết, khóa đã hết hạn, hay bất kỳ hành động nào làm thay đổi tập dữ liệu chính.

- Khi liên kết giữa bản gốc và bản sao bị hỏng, do sự cố mạng hoặc do thời gian chờ hết hạn, bản sao sẽ tự động kết nối lại và cố gắng tiến hành đồng bộ hóa lại một phần: có nghĩa là nó chỉ lấy lại phần nó bị mất trong quá trình kết nối.
- Khi không thể đồng bộ hóa một phần, bản sao sẽ yêu cầu đồng bộ hóa hoàn toàn. Điều này sẽ làm mất thời gian và phức tạp. Bởi nó là cả một quá trình, máy chủ sẽ cần tạo 1 ảnh chụp nhanh tất cả dữ liệu của nó, gửi nó tới bản sao và sau đó tiếp tục gửi dòng lệnh khi tập dữ liệu thay đổi.

#### 3.4.3.2. Đặc điểm Replication

- Redis sử dụng bản sao không đồng bộ, với dự thừa nhận từ bản sao đến chủ không đồng bộ về lượng dữ liệu được xử lý.
- Một master sẽ có nhiều bản sao.
- Bản sao có thể chấp nhận kết nối từ các bản sao khác. Ngoài việc kết nối một số bản sao với cùng một bản gốc, bản sao cũng có thể kết nối với bản sao khác trong cấu trúc giống như tầng.
- Bản gốc sẽ tiếp tục xử lý các truy vấn khi một hoặc nhiều bản sao thực hiện đồng bộ hóa ban đầu hoặc đồng bộ hóa một phần.

### 3.5. Ưu điểm và nhược điểm

Sau khi tìm hiểu về Redis, ta có thể rút ra ưu điểm và nhược điểm của nó.

#### 3.5.1. Ưu điểm

- Redis hỗ trợ thêm mới, cập nhật, xóa dữ liệu một cách nhanh chóng.
- Lưu trữ dữ liệu dạng khóa-giá trị.
- Dữ liệu được lưu trữ trên RAM giúp việc truy xuất dữ liệu một cách nhanh chóng. Ngoài ra, có thể cấu hình để Redis có thể lưu trữ dữ liệu trên ổ cứng.
- Tự cấu hình cho key tự động xóa trong khoảng thời gian nhất định.
- Hỗ trợ nhiều loại kiểu dữ liệu khác nhau.
- Hỗ trợ Queue (hàng đợi) thông qua cơ chế PUB/SUB, chúng ta có thể dùng Redis để làm hệ thống hàng đợi cho website xử lý tuần tự từng yêu cầu.

### 3.5.2. Nhược điểm

- Tính nhất quán dữ liệu thấp là nhược điểm chung của NoSQL
- Vì lưu trữ dữ liệu trên RAM, do đó không phù hợp để xử lý dữ liệu lớn.

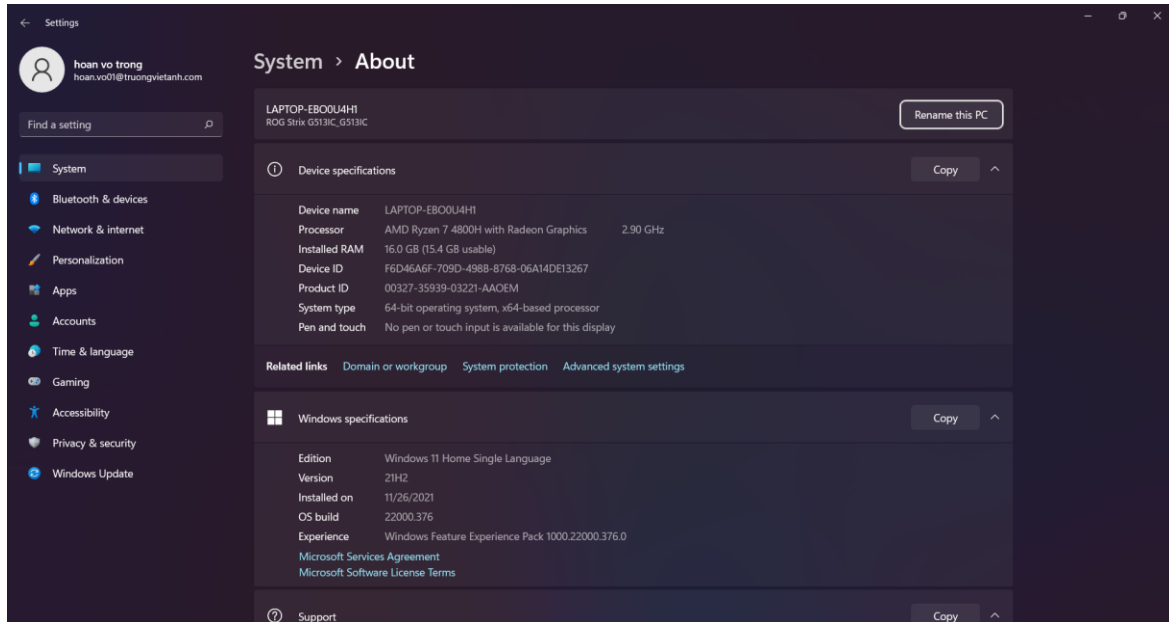
### 3.5.3. Ứng dụng

- Làm cache cho ứng dụng.
- Dùng để lưu thông tin trong sessions, profiles/preferences của người dùng.

## 4. CÀI ĐẶT VÀ THỰC NGHIỆM MÔ HÌNH PHÂN TÁN BẰNG REDIS

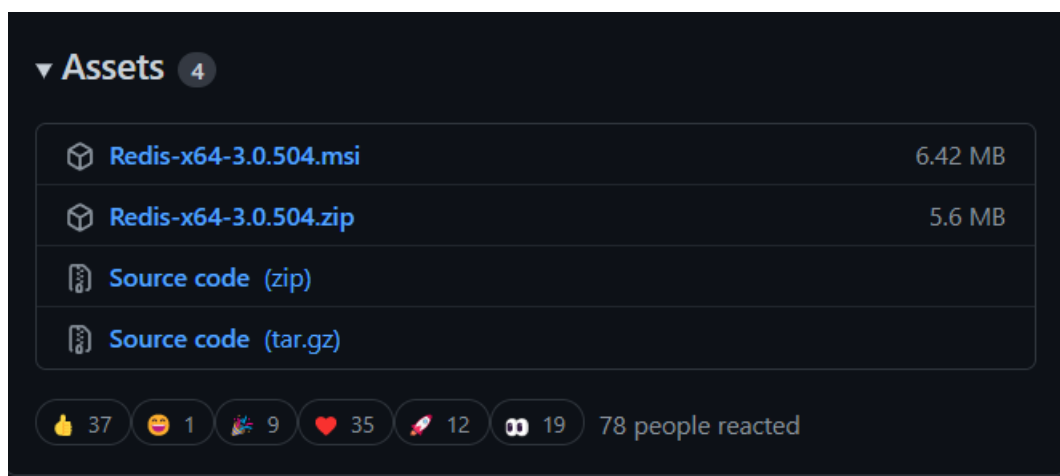
### 4.1. Cài đặt Redis

- Đầu tiên chúng ta cần kiểm tra cấu hình máy có thích hợp để cài đặt.



- Tải file cài đặt Redis trên Windows, chúng ta cần tải file cài đặt .msi hoặc file .zip (Redis chỉ hỗ trợ với win x64).

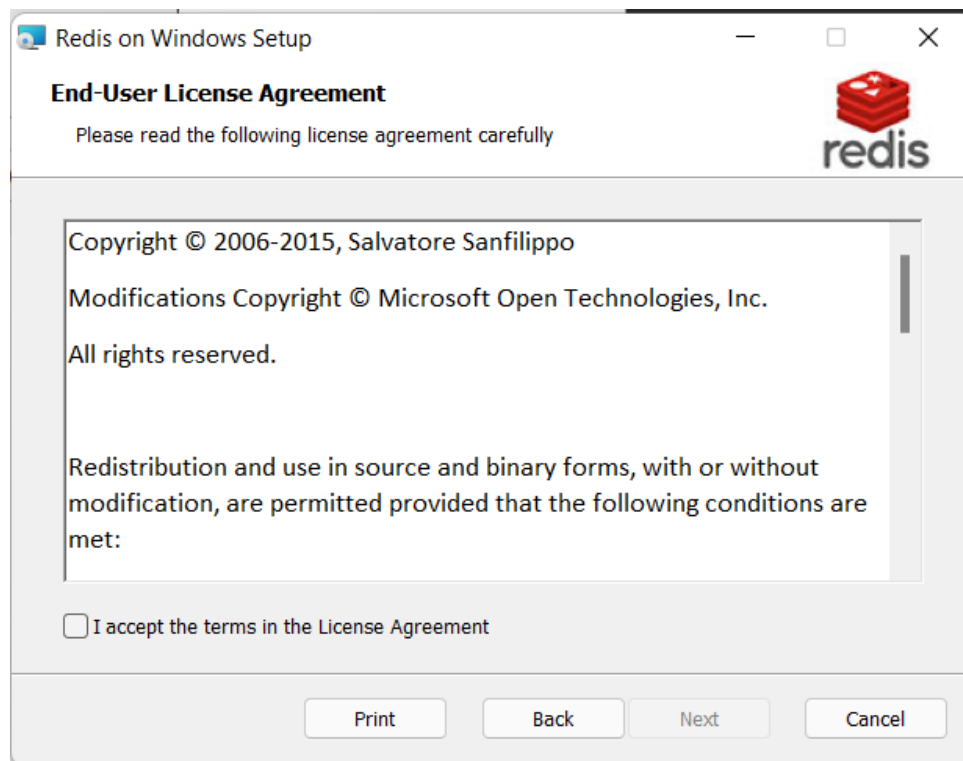
(<https://github.com/MicrosoftArchive/redis/releases>)



- Chạy file cài đặt vừa được tải theo các bước như sau:

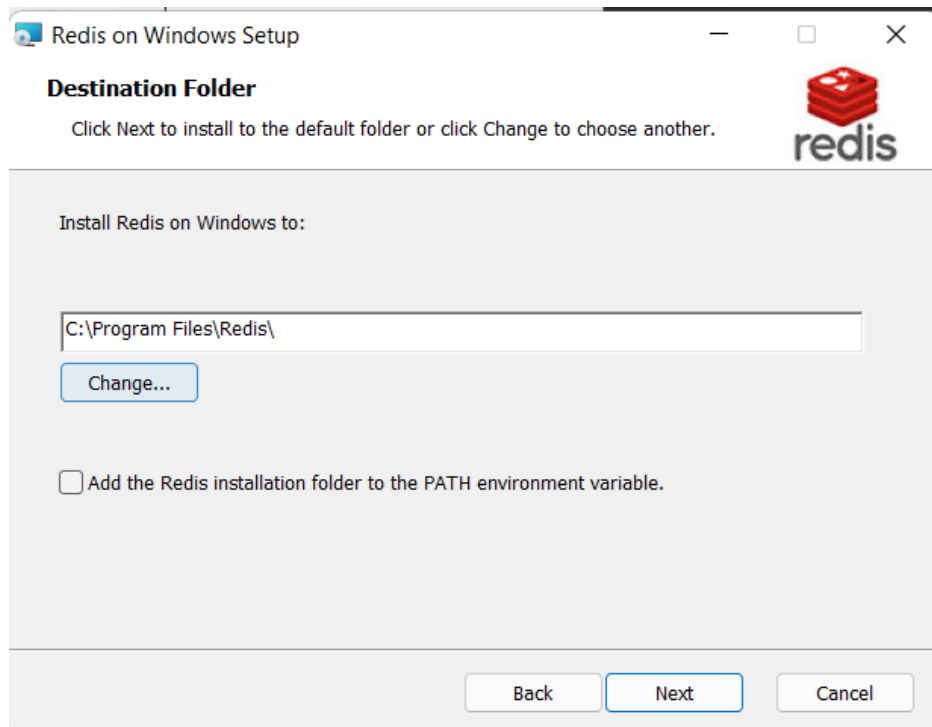


- Chọn Next để tiếp tục

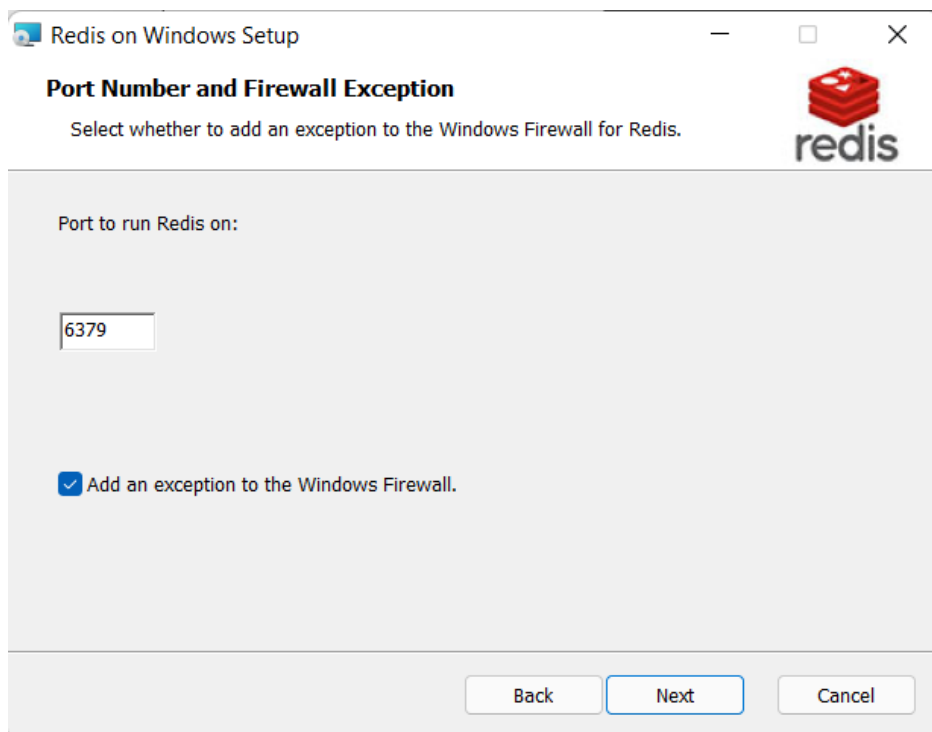




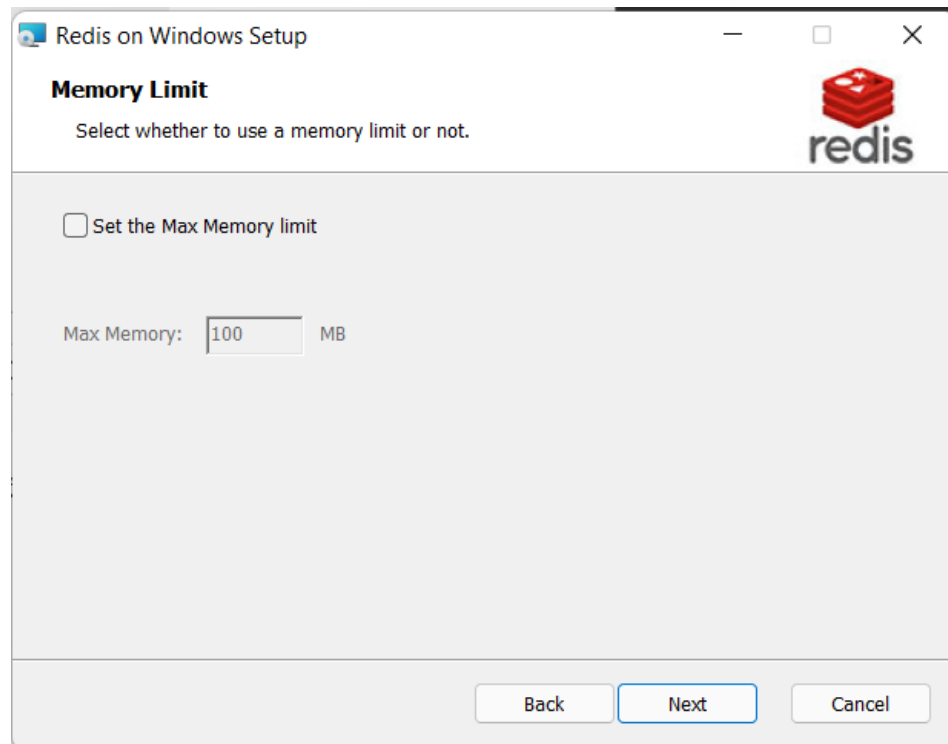
- Chọn “I accept the terms in the License Agreement” và sau đó bấm Next



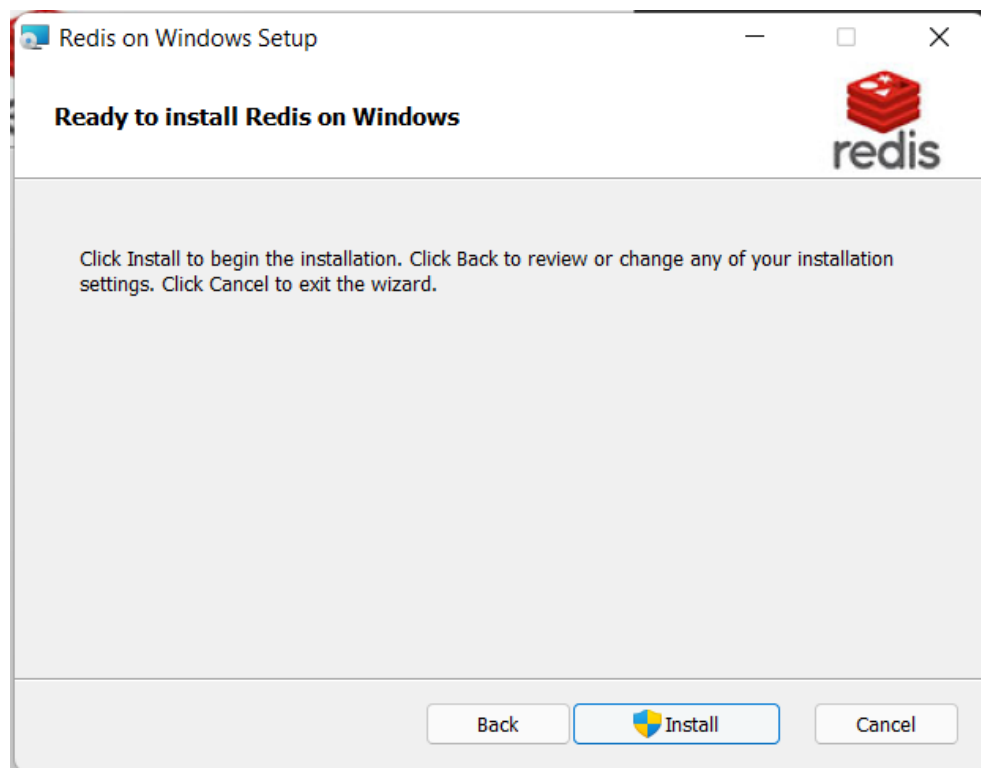
- Chọn đường dẫn đến ổ đĩa muốn lưu trữ và chọn vào “Add the Redis installation folder to the PATH environment variable”. Chọn Next



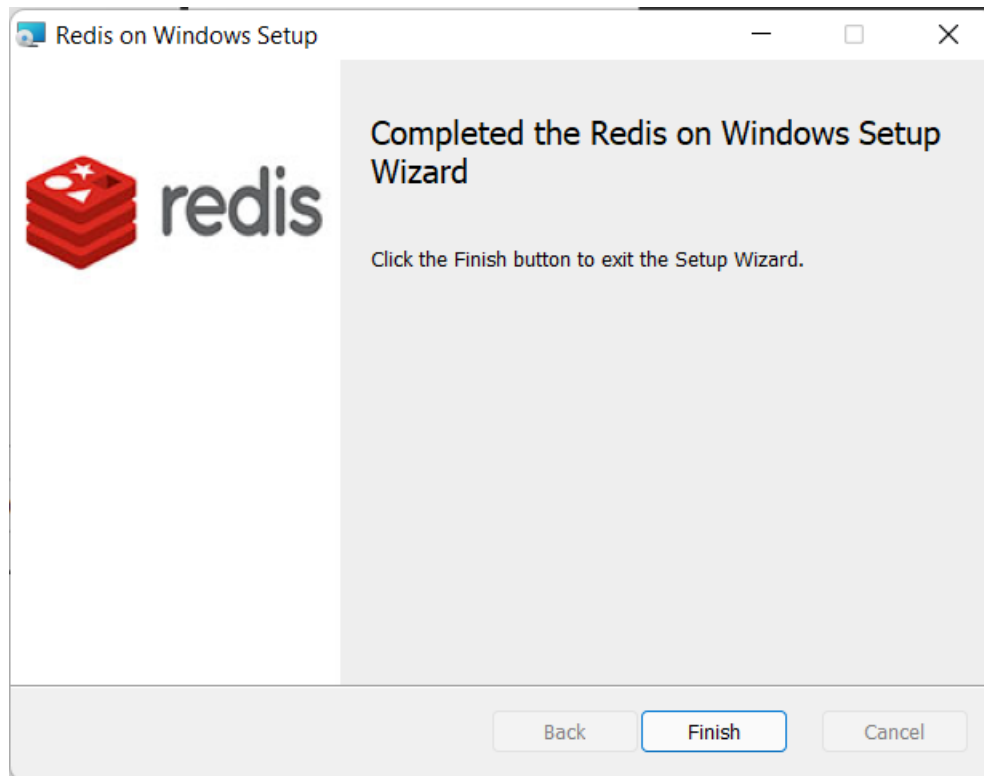
- Điền Port cho Redis hoặc dùng port mặc định. Sau đó chọn Next.



- Chọn Next.



- Chọn Install để bắt đầu cài đặt.



- Cuối cùng chọn Finish để kết thúc quá trình cài đặt.
- Mở cửa sổ cmd, chạy lệnh redis-cli để kiểm tra.

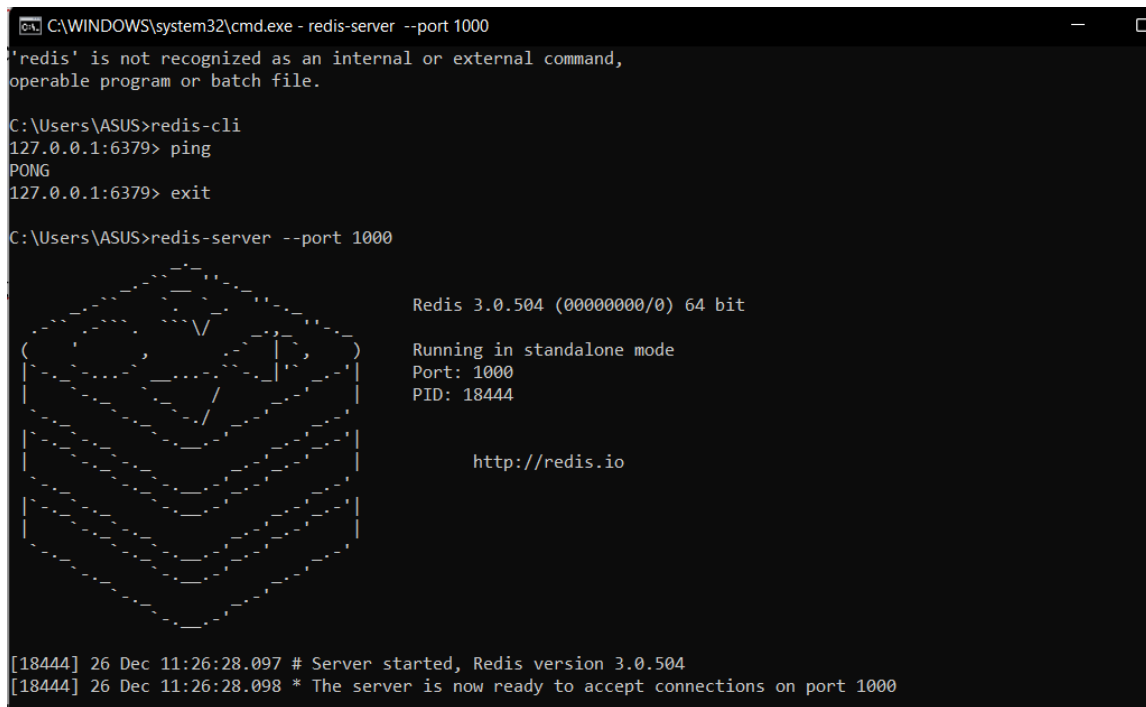
```
C:\Users\ASUS>redis-cli  
127.0.0.1:6379> █
```

- 127.0.0.1 là địa chỉ IP của máy.
  - 6379 là port mà Redis đang sử dụng.
- Sau đó chạy lệnh ping

```
C:\Users\ASUS>redis-cli  
127.0.0.1:6379> ping  
PONG  
127.0.0.1:6379> █
```

- Nếu trả về PONG thì redis đã cài đặt và chạy thành công.

- Để kiểm tra server của redis, exit và chạy lệnh `redis-server --port 1000`



```
C:\WINDOWS\system32\cmd.exe - redis-server --port 1000
'redis' is not recognized as an internal or external command,
operable program or batch file.

C:\Users\ASUS>redis-cli
127.0.0.1:6379> ping
PONG
127.0.0.1:6379> exit

C:\Users\ASUS>redis-server --port 1000

Redis 3.0.504 (00000000/0) 64 bit
Running in standalone mode
Port: 1000
PID: 18444

http://redis.io

[18444] 26 Dec 11:26:28.097 # Server started, Redis version 3.0.504
[18444] 26 Dec 11:26:28.098 * The server is now ready to accept connections on port 1000
```

Port 1000 là chạy server redis với port 1000 thay vì port mặc định.

#### 4.2. Thực hiện phân tán

- Tạo 1 cluster gồm 2 node với IP như sau:
  - Master 1: 26.224.114.67
  - Master 2: 26.167.87.71
- Sửa file **redis.windows.conf** trong thư mục cài đặt của Redis như sau:
  - Master 1:

```
bind 26.224.114.67
cluster-enabled yes
cluster-config-file nodes.conf
cluster-node-timeout 5000
appendonly yes
```

- Master 2:

```
bind 26.167.87.71  
  
cluster-enabled yes  
  
cluster-config-file nodes-5379.conf  
  
cluster-node-timeout 5000  
  
appendonly yes
```

***Ý nghĩa của từng tham số:***

- bind: là địa chỉ IP của máy host.
- cluster-enabled : cho phép redis chạy cluster mode.
- cluster-config-file: file chứa các cấu hình cho node này được lưu trữ, file conf mặc định là nodes-6379.conf. File này sẽ được tạo khi chạy cluster. Ở đây chúng ta sửa tên file để dễ phân biệt file của node.
- cluster-node-timeout: Thời gian timeout của node.
- appendonly: lưu trữ dữ liệu của Redis vào ổ đĩa AOF. File này sẽ được tạo khi Redis cluster chạy.

- Sau khi sửa xong ta mở Command Prompt chạy lại redis-server redis.windows.conf để khởi động server và mở 1 cửa sổ Command Prompt khác để kết nối.

```
D:\Redis>redis-server redis.windows.conf
[15296] 01 Jan 15:09:37.596 * Node configuration loaded, I'm 229c0a1eff5752709b19057a4ac10f9f84f6c831

Redis 3.2.100 (00000000/0) 64 bit

Running in cluster mode
Port: 6379
PID: 15296

http://redis.io

[15296] 01 Jan 15:09:37.596 # Server started, Redis version 3.2.100
[15296] 01 Jan 15:09:37.596 * DB loaded from append only file: 0.000 seconds
[15296] 01 Jan 15:09:37.596 * The server is now ready to accept connections on port 6379
```

- Tại máy Master 1 ta tiến hành mở redis-cli bằng lệnh

```
redis-cli -c -h 26.224.114.67
```

Trong đó:

- o -c <cluster>: cho phép chạy chế độ cluster.
- o -h <host>: địa chỉ ip của server host.
- o -p<port>: server port(mặc định là 6379).
- Tại Master 1
- Ta tiến hành kiểm tra số lượng node hiện tại bằng câu lệnh

```
cluster nodes
```

```
D:\Redis>redis-cli -c -h 26.224.114.67
26.224.114.67:6379> cluster nodes
229c0a1eff5752709b19057a4ac10f9f84f6c831 :6379 myself,master - 0 0 0 connected 2983
```

Ta thấy hiện tại chỉ có một node trong cluster. Tiến hành thêm các node làm master, ta thoát khỏi redis-cli và sử dụng lệnh cluster meet để thêm các node mới và giúp các node này giao tiếp với nhau.

- Thêm Master 2 vào cluster của Master 1
- Ta chạy lệnh

```
redis-cli -c -h 26.224.114.67 cluster meet 26.167.87.71 6379
```

```
D:\Redis>redis-cli -c -h 26.224.114.67 cluster meet 26.167.87.71 6379
OK
```

- Lúc này, ta kiểm tra lại số node trong cluster bằng lệnh **cluster nodes**

```
D:\Redis>redis-cli -c -h 26.224.114.67
26.224.114.67:6379> cluster nodes
bcd9c9c8033444fd865136dcac3c8d57c1565f1e 26.167.87.71:6379 master - 0 1641024910305 0 connected
229c0a1eff5752709b19057a4ac10f9f84f6c831 26.224.114.67:6379 myself,master - 0 0 1 connected 2983
```

Ta thấy node Master 2 đã được thêm vào chung 1 cluster. Tuy nhiên, những node này chưa được gán cho các slot là những không gian lưu trữ

- Ta tiến hành phân chia không gian lưu trữ cho các node master như sau:
  - Slot từ 0-5460 sẽ được chia cho Master 1
  - Slot 5461-10922 sẽ được chia cho Master 2
- Redis chỉ hỗ trợ hàm ADDSLOTS cho phép chia từng slot vào các node. Vì vậy để chia số lượng lớn các slot vào các node ta sẽ sử dụng vòng trên Powershell. Tiến hành tạo một file powershell có nội dung như sau:

```
Windows PowerShell
Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\ASUS> cd "D:\Redis"
PS D:\Redis> Write-Host "Assign slot for master 1" -ForegroundColor yellow -BackgroundColor black
Assign slot for master 1
PS D:\Redis> for($slot =0; $slot -le 5460; $slot++){
>> Write-Progress -Activity "Assign slot for master 1" -Status "Assigning slot $slot / 5460:" -PercentComplete ($slot / 5460 *100);
>> [void](.\redis-cli.exe -h 26.224.114.67 CLUSTER ADDSLOTS $slot)
>> }
PS D:\Redis>
PS D:\Redis> Write-Host "Assign slot for master 2" -ForegroundColor yellow -BackgroundColor black
Assign slot for master 2
PS D:\Redis> for($slot =5461; $slot -le 10922; $slot++){
>> Write-Progress -Activity "Assign slot for master 2" -Status "Assigning slot $slot / 10922:" -PercentComplete ($slot / 10922 *100);
>> [void](.\redis-cli.exe -h 26.167.87.71 CLUSTER ADDSLOTS $slot)
>> }
PS D:\Redis>
```

- Sau khi thực thi trên PowerShell thì chúng ta được kết quả như sau

```
D:\Redis>redis-cli -c -h 26.224.114.67
26.224.114.67:6379> cluster nodes
bcd9c9c8033444fd865136dcac3c8d57c1565f1e 26.167.87.71:6379 master,fail? - 1641027863598 1641026636461 0 connected 5461-10922
229c0a1eff5752709b19057a4ac10f9f84f6c831 26.224.114.67:6379 myself,master - 0 0 1 connected 0-5460
26.224.114.67:6379>
```

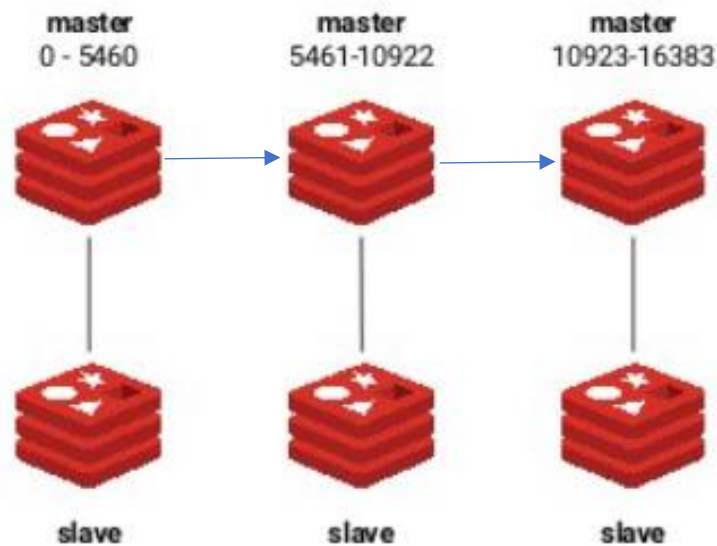
Ta có thể thấy slot đã được chia như mong muốn.

### 4.3. Thực nghiệm mô hình phân tán

#### 4.3.1. Mô tả bài toán

Đề bài: Quản lí chi nhánh của cửa hàng Deadline

Mô tả: Hệ thống cửa hàng có cửa hàng chính tại Hồ Chí Minh. Hiện nay, ngoài cửa hàng chính, hệ thống còn có chi nhánh tại Đà Nẵng và Hà Nội. Người ta tiến hành sử dụng Redis để dung dụng bộ nhớ đệm để giảm độ trễ, tăng hiệu suất và giảm tải cho cơ sở dữ liệu quan hệ và ứng dụng. Mô hình được xây dựng như sau:



Dữ liệu của hệ thống cửa hàng lưu trữ trong Redis được mô tả như sau:

**NHANVIEN (MANV, HOTEN, NGAYSINH, GIOITINH, CMND, CHUCVU, MACH, NGAYVAOLAM)**

Kiểu dữ liệu lưu trữ: Mỗi nhân viên sẽ được lưu trữ trong một HASH

Mô tả: Mỗi key - value NHANVIEN nhằm mô tả thông tin của một nhân viên của cửa hàng. Mỗi nhân viên bao gồm các thông tin mã nhân viên (MANV), họ và tên nhân viên (HOTEN), ngày sinh (NGAYSINH), giới tính (GIOITINH), số chứng minh thư nhân dân (CMND), chức vụ của nhân viên (CHUCVU), mã số cửa hàng mà nhân viên đó làm việc (MACH), ngày nhân viên được nhận vào làm (NGAYVAOLAM).



## **CUAHANG (MACH, TENCH, CHINHANH)**

Kiểu dữ liệu lưu trữ: Mỗi cửa hàng sẽ được lưu trữ trong một HASH

Mô tả: Mỗi key - value CUAHANG nhằm mô tả thông tin của tất cả các cửa hàng trong hệ thống. Thông tin được ghi nhận bao gồm: mã cửa hàng (MACH), tên cửa hàng (TENCH) và chi nhánh của cửa hàng đó (CHINHANH).

## **SANPHAM (MASP, TENSP, SOLUONG, DONGIA, NGAYNHAP, MACH)**

Kiểu dữ liệu lưu trữ: Mỗi nhạc cụ sẽ được lưu trữ trong một HASH

Mô tả: Mỗi key - value SANPHAM nhằm mô tả thông tin chi tiết của các sản phẩm hiện được bày bán ở mỗi cửa hàng. Các thông tin được lưu trữ bao gồm: mã sản phẩm (MASP), tên sản phẩm (TENSP), số lượng hiện có trong cửa hàng (SOLUONG), đơn giá (DONGIA), ngày lô sản phẩm được nhập về cửa hàng (NGAYNHAP), mã cửa hàng hiện đang bày bán sản phẩm (MACH).

### 4.3.2. Các bước thực nghiệm

Tại một master bất kỳ trong cluster, ta sẽ tiến hành thêm dữ liệu vào các master. Ở đây ta sẽ sử dụng master 1. Khi thêm dữ liệu vào, Redis sẽ chia dữ liệu vào các node master trong cluster.

Thêm CH01 bằng lệnh CLUSTER KEYSLOT

```
D:\cluster-redis\7000>redis-cli -c -p 7000
127.0.0.1:7000> cluster keyslot CuaHang:CH01
(integer) 3196
```

Ta thấy được key CuaHang:CH01 đã được lưu vào slot 3196. Và các dữ liệu của key CH01 sẽ được lưu vào slot 3196

```
127.0.0.1:7000> HMSET CuaHang:CH01 MaCuaHang CH01 TenCuaHang 'Cua Hang Deadline' DiaChi 'HCM'
OK
```

Ta tiến hành lưu dữ liệu lần lượt của Cửa hàng, Nhân viên, Sản phẩm:

```
127.0.0.1:7000> HMSET CuaHang:CH01 MaCuaHang CH01 TenCuaHang 'Cua Hang Deadline' DiaChi 'HCM'
OK
127.0.0.1:7000> HMSET CuaHang:CH02 MaCuaHang CH02 TenCuaHang 'Cua Hang Deadline chi nhanh 2' DiaChi 'Da Nang'
-> Redirected to slot [15391] located at 127.0.0.1:7002
OK
127.0.0.1:7002> HMSET CuaHang:CH03 MaCuaHang CH03 TenCuaHang 'Cua Hang Deadline chi nhanh 3' DiaChi 'Ha Noi'
OK
```

```
127.0.0.1:7002> HMSET NhanVien: NV01 HoTen'Vo Trong Hoan' NgaySinh '24/05/2001' GioiTinh 'Nam' CMND '123456' ChucVu 'Quan Ly' MaCH 'CH01' NgayVaoLam '01/01/2021'
-> Redirected to slot [5817] located at 127.0.0.1:7001
OK
127.0.0.1:7001> HMSET NhanVien: NV02 HoTen'Nguyen Thi Thuy Nga' NgaySinh '24/07/2001' GioiTinh 'Nu' CMND '1234567' ChucVu 'Quan Ly' MaCH 'CH21' NgayVaoLam '01/01/2021'
OK
127.0.0.1:7001> HMSET NhanVien: NV03 HoTen'Nguyen Quang Huy' NgaySinh '25/07/2001' GioiTinh 'Nam' CMND '1234567' ChucVu 'Quan Ly' MaCH 'CH3' NgayVaoLam '01/01/2021'
OK
```

```
127.0.0.1:7001> HMSET SanPham:SP01 MaSP 'SP01' TenSP 'Vo o ly' SoLuong '100' DonGia '10000' NgayNhap '01/01/2022' MaCH 'CH01'
-> Redirected to slot [14177] located at 127.0.0.1:7002
OK
127.0.0.1:7002> HMSET SanPham:SP01 MaSP 'SP01' TenSP 'Vo o ly' SoLuong '100' DonGia '10000' NgayNhap '01/01/2022' MaCH 'CH2'
OK
127.0.0.1:7002> HMSET SanPham:SP01 MaSP 'SP01' TenSP 'Vo o ly' SoLuong '100' DonGia '10000' NgayNhap '01/01/2022' MaCH 'CH03'
OK
```

Ta tiến hành truy vấn ở một số node:

Master 1:

```
127.0.0.1:7002> HGETALL CuaHang:CH01
-> Redirected to slot [3196] located at 127.0.0.1:7000
1) "MaCuaHang"
2) "CH01"
3) "TenCuaHang"
4) "Cua Hang Deadline"
5) "DiaChi"
6) "HCM"
```

```
127.0.0.1:7002> HMGET NhanVien:NV1 MaNV HoTen
1) "NV1"
2) "Vo Trong Hoan"
127.0.0.1:7002> _
```

Master 2:

```
D:\cluster-redis\7001>redis-cli -c -p 7001
127.0.0.1:7001> HGETALL CuaHang:CH03
-> Redirected to slot [11326] located at 127.0.0.1:7002
1) "MaCuaHang"
2) "CH03"
3) "TenCuaHang"
4) "Cua Hang Deadline chi nhanh 3"
5) "DiaChi"
6) "Ha Noi"
127.0.0.1:7002> HGETALL:SP01
(error) ERR unknown command 'HGETALL:SP01'
127.0.0.1:7002> HGETALL SanPham:SP01
1) "MaSP"
2) "SP01"
3) "TenSP"
4) "Vo o ly"
5) "SoLuong"
6) "100"
7) "DonGia"
8) "10000"
9) "NgayNhap"
10) "01/01/2022"
11) "MaCH"
12) "CH03"
127.0.0.1:7002>
```

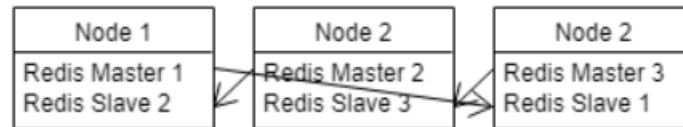
Slave 1:

```
127.0.0.1:7003> HGETALL CuaHang:CH03
-> Redirected to slot [11326] located at 127.0.0.1:7002
1) "MaCuaHang"
2) "CH03"
3) "TenCuaHang"
4) "Cua Hang Deadline chi nhanh 3"
5) "DiaChi"
6) "Ha Noi"
```

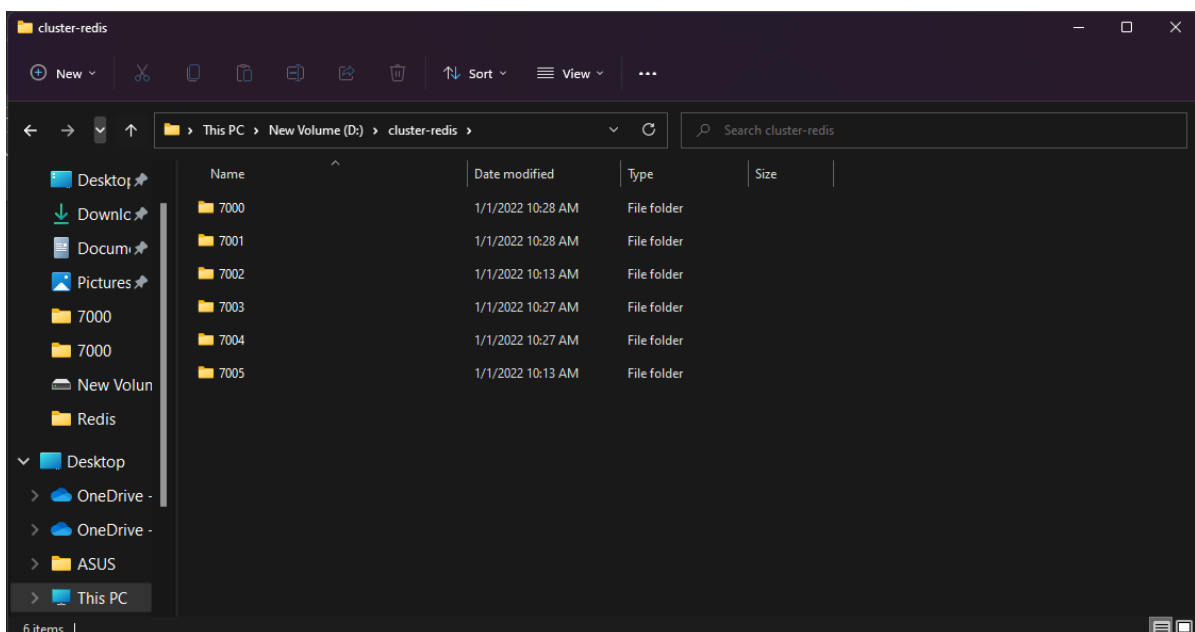
```
127.0.0.1:7000> HMGET NhanVien:NV1 MaNV HoTen
-> Redirected to slot [14182] located at 127.0.0.1:7002
1) "NV1"
2) "Vo Trong Hoan"
```

## 5. CƠ CHẾ NHÂN BẢN

Nhân bản trong Redis chúng ta sử dụng cơ chế Master-Slave.



- Ở đây chúng ta sử dụng 6 node tương ứng với 6 port trong máy:
  - Master 1: 7000
  - Master 2: 7001
  - Master 3: 7002
  - Slave 1: 7003
  - Slave 2: 7004
  - Slave 3: 7005
- Chúng ta sẽ tạo mỗi node 1 thư mục tương ứng với tên port và lưu trong 1 thư mục chung



- Trong mỗi thư mục cài đặt của mỗi node, ta sửa file redis.windows.conf ở những mục sau:

```
port 7000

cluster-enabled yes

cluster-config-file nodes-7000.conf

cluster-node-timeout 5000

appendonly yes
```

- Tương tự với các master và slave còn lại tương ứng theo port.
- Sau đó, tải Ruby tại link <https://rubyinstaller.org/downloads/>
- Chạy file cài đặt trên máy.
- Tải Rubigems tại link <https://rubygems.org/pages/download> ở đây ta tải file zip của rubygems. Tiến hành giải nén, và chuyển cửa sổ CMD qua cửa sổ đã giải nén và chạy lệnh: `ruby setup.rb`
- Chuyển cửa sổ về thư mục chứa các node và chạy lệnh `gem install redis`
- Khi thành công sẽ in ra:

```
D:\cluster-redis>gem install redis

Fetching redis-4.1.3.gem
Successfully installed redis-4.1.3
Parsing documentation for redis-4.1.3
Installing ri documentation for redis-4.1.3
Done installing documentation for redis after 1 seconds
1 gem installed
```

- Tải về tệp tin redis-trib.rb qua link <https://github.com/beeol/redis-trib.rb> và đặt nó vào thư mục của chứa các node.

- Sau đó tạo cụm cluster , chuyển cửa sổ cmd đến thư mục chứa các node và chạy lệnh:

```
ruby redis-trib.rb create --replicas 1 127.0.0.1:7000
127.0.0.1:7001 127.0.0.1:7002 127.0.0.1:7003 127.0.0.1:7004
127.0.0.1:7005
```

- Khi tạo thành công thì chúng ta sẽ có màn hình như sau:

```
>>> Creating cluster
>>> Performing hash slots allocation on 6 nodes...
Using 3 masters:
127.0.0.1:7000
127.0.0.1:7001
127.0.0.1:7002
Adding replica 127.0.0.1:7003 to 127.0.0.1:7000
Adding replica 127.0.0.1:7004 to 127.0.0.1:7001
Adding replica 127.0.0.1:7005 to 127.0.0.1:7002
M: 4f91aaa7bdd34a04b523811b94c5d2ceeac08bca 127.0.0.1:7000
slots:0-5460 (5461 slots) master
M: 16f0c79b9e15c018bfcac6132ec491bd3f4a2556 127.0.0.1:7001
slots:5461-10922 (5462 slots) master
M: b7c7139c2a5d730a45d94eef4ec356ae7212e0d9 127.0.0.1:7002
slots:10923-16383 (5461 slots) master
S: 38ccafe08b04c720e180898559816ec14eea5dd0 127.0.0.1:7003
replicates 4f91aaa7bdd34a04b523811b94c5d2ceeac08bca
S: 0f5615c3c3aba768221f32c0ec08d2f6c9ca78f2 127.0.0.1:7004
replicates 16f0c79b9e15c018bfcac6132ec491bd3f4a2556
S: f544e9d36b26d8dc870cb38ffcc8d8a32bdbeb1d 127.0.0.1:7005
replicates b7c7139c2a5d730a45d94eef4ec356ae7212e0d9
Can I set the above configuration? (type 'yes' to accept): yes
>>> Nodes configuration updated
>>> Assign a different config epoch to each node
>>> Sending CLUSTER MEET messages to join the cluster
Waiting for the cluster to join...
>>> Performing Cluster Check (using node 127.0.0.1:7000)
M: 4f91aaa7bdd34a04b523811b94c5d2ceeac08bca 127.0.0.1:7000
slots:0-5460 (5461 slots) master
M: 16f0c79b9e15c018bfcac6132ec491bd3f4a2556 127.0.0.1:7001
slots:5461-10922 (5462 slots) master
M: b7c7139c2a5d730a45d94eef4ec356ae7212e0d9 127.0.0.1:7002
slots:10923-16383 (5461 slots) master
M: 38ccafe08b04c720e180898559816ec14eea5dd0 127.0.0.1:7003
slots: (0 slots) master
replicates 4f91aaa7bdd34a04b523811b94c5d2ceeac08bca
M: 0f5615c3c3aba768221f32c0ec08d2f6c9ca78f2 127.0.0.1:7004
slots: (0 slots) master
replicates 16f0c79b9e15c018bfcac6132ec491bd3f4a2556
M: f544e9d36b26d8dc870cb38ffcc8d8a32bdbeb1d 127.0.0.1:7005
slots: (0 slots) master
replicates b7c7139c2a5d730a45d94eef4ec356ae7212e0d9
[OK] All nodes agree about slots configuration.
>>> Check for open slots...
>>> Check slots coverage...
[OK] All 16384 slots covered.
```

Cơ chế:

Các Master sẽ tương tác với data, Slave chỉ có quyền xem. Vì vậy các Master sẽ được phân vùng nhớ (slot).

Khi 1 Master bị lỗi thì slave sẽ đưa lên làm Master và thừa hưởng vùng nhớ đó của Master.

## GITHUB

[https://github.com/uit19521881/phantan\\_IS211.M11.HTCL\\_7](https://github.com/uit19521881/phantan_IS211.M11.HTCL_7)

## PHÂN CÔNG CÔNG VIỆC

Công việc	Thúy Nga	Trọng Hoàn	Quang Huy
Tìm tài liệu tham khảo	60%	20%	20%
Tìm hiểu NoSQL	70%	15%	15%
Tìm hiểu Redis	15%	70%	15%
Tìm hiểu cách cài đặt Redis và cơ chế phân tán trên Redis	15%	15%	70%
Viết báo cáo	33%	33%	33%
Định dạng báo cáo	100%		
Thiết kế slide	40%	30%	30%



## TÀI LIỆU THAM KHẢO

1. <https://nosql-database.org/>
2. <https://redis.io/documentation>
3. <http://www.yayihouse.com/yayishuwu/chapter/2416>